

Fall 1-16-2014

## Mobile Robot Localization Based on Kalman Filter

Omar Q. Mohsin  
*Portland State University*

Let us know how access to this document benefits you.

Follow this and additional works at: [http://pdxscholar.library.pdx.edu/open\\_access\\_etds](http://pdxscholar.library.pdx.edu/open_access_etds)



Part of the [Robotics Commons](#)

---

### Recommended Citation

Mohsin, Omar Q., "Mobile Robot Localization Based on Kalman Filter" (2014). *Dissertations and Theses*. Paper 1529.

[10.15760/etd.1528](https://doi.org/10.15760/etd.1528)

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. For more information, please contact [pdxscholar@pdx.edu](mailto:pdxscholar@pdx.edu).

Mobile Robot Localization Based on Kalman Filter

by

Omar Q. Mohsin

A thesis submitted in partial fulfillment of the  
requirements for the degree of

Master of Science  
in  
Electrical and Computer Engineering

Thesis Committee:  
Marek A. Perkowski, Chair  
Douglas V. Hall  
Xiaoyu Song

Portland State University  
2013

© 2013 Omar Q. Mohsin

## ABSTRACT

Robot localization is one of the most important subjects in the Robotics science. It is an interesting and complicated topic. There are many algorithms to solve the problem of localization. Each localization system has its own set of features, and based on them, a solution will be chosen. In my thesis, I want to present a solution to find the best estimate for a robot position in certain space for which a map is available. The thesis started with an elementary introduction to the probability and the Gaussian theories. Simple and advanced practical examples are presented to illustrate each concept related to localization. Extended Kalman Filter is chosen to be the main algorithm to find the best estimate of the robot position. It was presented through two chapters with many examples. All these examples were simulated in Matlab in this thesis in order to give the readers and future students a clear and complete introduction to Kalman Filter.

Fortunately, I applied this algorithm on a robot that I have built its base from scratch. MCECS-Bot was a project started in Winter 2012 and it was assigned to me from my adviser, Dr. Marek Perkowski. This robot consists of the base with four Mecanum wheels, the waist based on four linear actuators, an arm, neck and head. The base is equipped with many sensors, which are bumper switches, encoders, sonars, LRF and Kinect. Additional devices can provide extra information as backup sensors, which are a tablet and a camera. The ultimate goal of this thesis is to have the MCECS-Bot as an open source system accessed by many future classes, capstone projects and graduate thesis students for education purposes.



A well-known MRPT software system was used to present the results of the Extended Kalman Filter (EKF). These results are simply the robot positions estimated by EKF. They are demonstrated on the base floor of the FAB building of PSU. In parallel, simulated results to all different solutions derived in this thesis are presented using Matlab. A future students will have a ready platform and a good start to continue developing this system.

## ACKNOWLEDGEMENTS

I take this opportunity to thank all people who helped me in many ways in bringing out this thesis.

Primarily, I would like to express my deep gratitude to my advisor Dr. Marek Perkowski for his incessant support, encouragement, guidance and advice that made this thesis possible. His emphasis on the quality of research and writing has been extremely valuable in writing this thesis.

I would like to thank the Dean of Engineering at PSU for supporting the project's team financially and for being kind and supportive during our meetings. I would like to thank my colleague Mathias Sunardi for his help and support. Without his help, I would not have gotten the results using the MRPT. I thank all my friends in Portland for their support and belief in me to accomplish this project.

My gratitude is to my mother for her infinite patience and implicit faith in my capabilities, and her love as well as her caring is boundless and cannot be expressed in sufficient words. Additionally, I would like to truly express my deepest gratitude to my father who passed away three months before I came to United States. He was always encouraging me to obtain a Masters and PHD. To my father, I wish you are here sharing this happiness with me and I would like to tell him that I fulfilled part of my promise. I feel that my father is also looking at this work from Heaven. I am extremely grateful to my wife, Marisela, for being patient and to my little daughter, Yasmine, for giving me happiness and making me feel good whenever being under stress. My gratitude is going

also to my friends and relatives who are back home. They never stopped calling to strengthen me with their encouraging words.

Special thanks to Fulbright scholarship program, which without them in the first place, I would not be here. Everyone in this program was supportive and ready to help. They have made many obstacles easier for me in order to achieve this dream.

## Table of Contents

ABSTRACT.....	i
ACKNOWLEDGEMENTS.....	iii
List of Tables.....	ix
List of Figures.....	x
Chapter 1: Introduction.....	3
Chapter 2: Mathematical Introduction:.....	12
2.1 Permutation.....	12
2.1.1 Permutation with Repetition.....	12
2.1.2 Permutation without Repetition.....	13
2.1.3 Example.....	13
2.2 Combination.....	13
2.2.1 Combinations without Repetition.....	14
2.3 Standard Deviation.....	14
2.3.1 Variance and Standard Deviation Example [2].....	15
2.4 Expected Value.....	17
2.4.1 Example.....	17
2.4.2 Example.....	18
2.5 Probability.....	19
2.5.1 General Rules.....	19
2.5.2 Simple Examples of using Probability.....	20
2.5.3 Binomial Probability (Discrete).....	24
2.5.4 Probability Distribution Function.....	26
2.5.5 Expected Value of Binomial Distribution.....	27
2.5.6 Conditional Probability.....	28
2.5.7 Bayes' Theorem.....	29
2.6 Variance.....	30
2.6.1 Variance of Binomial Distribution.....	33
2.7 Random (or Stochastic) Variable.....	34
2.7.1 Discrete Random Variables.....	34
2.7.2 Continuous Random Variables.....	35
2.8 Normal (or Gaussian) Distribution (Continuous).....	37
2.8.1 Example.....	39
2.8.2 Example.....	41
2.9 Normal Distribution and Binomial Distribution.....	41
2.10 Compare Normal distribution to other Distribution Functions.....	43
2.11 Normal Distribution Empirical Rule.....	45
2.11.1 Example.....	46
Chapter 3: Introduction to Kalman Filter.....	49
3.1 Average Filter.....	49
3.1.1 Example.....	50
3.2 Moving Average Filter (MAF).....	54
3.2.1 Example.....	54

3.3	Low Pass Filter (LPF)	58
3.3.1	Example	59
Chapter 4: Kalman Filter		64
4.1	A Comparison between Kalman Filter and Low Pass Filter	66
4.2	Error Covariance	68
4.3	Prediction Process	70
4.4	State Space Representation	72
4.5	State Space Property	75
4.6	Writing a System in a State Space Form Requires:	76
4.7	State Space Model Examples	77
4.7.1	Example	77
4.7.2	Example	78
4.8	Kalman Filter State Model	79
4.9	Covariance Matrix	80
4.10	The Effect of Matrix R and Q	81
4.11	Kalman Filter Examples	82
4.11.1	One Dimension Kalman Example	82
4.11.2	Estimating Velocity from Position	87
4.11.2.1	First case	88
4.11.3	Two Dimension Kalman Filter	99
4.11.4	Changing Matrices $Q$ and $R$	101
4.12	Sensor Fusion	103
4.12.1	Using Gyro to get Roll and Pitch angles	106
4.12.2	Using Accelerometers to get Roll and Pitch Angles	111
4.12.3	Using Sensor Fusion	116
4.13	Extended Kalman Filter	124
4.13.1	Altitude Reference System Example	126
Chapter 5: MCECS Guide Bot		134
5.1	Base	139
5.1.1	Mechanical Design and Basic Power Calculations	140
5.1.2	Robot Design	144
5.1.3	Construction Parts	146
5.1.3.2	Bolts	147
5.1.4	Base Parts	152
5.1.5	Actual Work	179
5.2	Waist	183
5.3	Arms	184
5.4	Neck	185
5.5	Head	185
5.6	Software System	187
5.6.1	Overview	188
5.6.2	The Server Program: Server.py	190
5.6.3	Kinect	192
5.6.4	MRPT Code (Navigation Program):	193

5.7	Overall Circuit Connections.....	195
5.8	Prices .....	197
5.8.1	Base.....	197
5.8.2	Waist.....	201
Chapter 6:	Literature Review .....	204
6.1	Mecanum Wheels.....	204
6.2	Localization and Sensor Fusion .....	207
6.3	Other Guide and Museum Robots.....	210
6.3.1	RHINO [48] .....	211
6.3.2	Minerva [49] .....	211
6.3.3	RoboX [50] .....	212
6.3.4	Maggie [51].....	213
6.3.5	Comparison between the Presented Robots and MCECS-Bot .....	214
Chapter 7:	Using Kalman Filter with MCECS-BOT .....	216
7.1	Wall Following Algorithm .....	217
7.1.1	Normal Wall following .....	217
7.1.2	Wall Following with Kalman Filter .....	220
7.2	Localization Based on Kalman Filter.....	228
7.2.1	One-Dimensional Extended Kalman Filter.....	228
7.2.2	Two-Dimensional Extended Kalman Filter .....	248
Chapter 8:	Conclusion, Future Work and My Contribution.....	275
8.1	Conclusion.....	275
8.2	Future Work.....	278
8.2.1	Sonars.....	278
8.2.2	Encoders.....	279
8.2.3	Battery and Velocity Indicator .....	280
8.2.4	Advanced Localization based on Extended Kalman Filter.....	281
8.2.5	Connecting the MCECS-Bot to the Internet .....	285
8.2.6	Testing for a Long Time Operation.....	286
8.3	My Contributions in this Thesis .....	286
References	.....	288
Appendix A:	Practicle Applications for Mobile Robots based on Mecanum Wheels- a Systematic Survey [19] .....	293
Appendix B:	RoboClaw 2 Channel 15A Motor Controller Data Sheet [52] .....	308
Appendix C:	Motor Specifications [25].....	369
Appendix D:	Encoder Specifications [26] .....	370
Appendix E:	Battery .....	374
	Battery Specifications [53].....	374
	Material Safety Data Sheet [54].....	376
Appendix F:	System Circuit Boards Diagram .....	382
Appendix G:	MB1000 LV-MaxSonar®-EZ0™ [29] .....	388
Appendix H:	The LiFePo4 Battery Charger [37].....	390
Appendix I:	Code.....	393
	Arduino Uno Code .....	393

Arduino Mega_1 Code .....	403
MRPT Code.....	439

## List of Tables

Table 4.1: Kalman Filter variables.....	65
Table 4.2: Q and R matrices.....	81
Table 5.1: Conventional vs. Omni wheels. ....	152
Table 5.2: Omni vs. Mecanum wheels.....	153
Table 5.3: Mecanum wheel advantages and disadvantages.....	154
Table 5.4: DC Motor specification.....	157
Table 5.5: Arduino specifications. ....	162
Table 5.6: Different Wheels characteristics with their prices. ....	197
Table 5.7: Different gearbox with their prices. ....	198
Table 5.8: Different motor drivers' prices with specifications.....	199
Table 5.9: Arduino boards prices that used in the base.....	199
Table 5.10: Hardware prices. ....	200
Table 5.11: Parts ordered online.....	201
Table 5.12: Hardware parts purchased from local hardware stores. ....	202
Table 6.1: A comparison between MCECS-Bot and other guide robots.....	214



## List of Figures

Figure 1.1: PEOPLE-Bot. ....	5
Figure 1.2: MCECS-Bot (PSU Guide Robot) [2]. ....	6
Figure 2.1: Standard deviation example. ....	15
Figure 2.2: Standard deviation example. ....	16
Figure 2.3: Probability illustration. ....	19
Figure 2.4: Probability diagram. ....	21
Figure 2.5: The distribution of coins. ....	21
Figure 2.6: Binomial Distribution. ....	24
Figure 2.7: Raining probability. ....	25
Figure 2.8: Discrete and continuous probability distribution function examples [17]. ....	26
Figure 2.9: Probability distribution function example. ....	26
Figure 2.10: Bag filled with 3 white and 2 black marbles. ....	28
Figure 2.11: Jar filled with stars and marbles. ....	29
Figure 2.12: Variance for a population and a sample of a population. ....	32
Figure 2.13: The probability distribution of all possible outcomes is discrete random values. ....	35
Figure 2.14: Continuous values representing the amount of raining. ....	35
Figure 2.15: Finding the area between two sets (finding the probability of a range). ....	36
Figure 2.16: Finding the probability below a certain value. ....	37
Figure 2.17: Normal distribution and Cumulative distribution function examples. ....	38
Figure 2.18: The normal distribution. ....	40
Figure 2.19: The cumulative distribution function. ....	41
Figure 2.20: Binomial distribution. ....	42
Figure 2.21: Normal distribution. ....	43
Figure 2.22: Double humped normal distribution. ....	44
Figure 2.23: Positive skew distribution. ....	44
Figure 2.24: Negative skewed distribution. ....	45
Figure 2.25: 68 95 99.7 rule [10]. ....	45
Figure 2.26: Normal distribution. ....	47
Figure 3.1: Average filter equation. ....	51
Figure 3.2: Get Sonar. ....	52
Figure 3.3: Test Average filter. ....	53
Figure 3.4: Average filter results. ....	53
Figure 3.5: Moving Average Filter code. ....	55
Figure 3.6: Get Sonar code. ....	56
Figure 3.7: Test Average filter code. ....	57
Figure 3.8: Moving Average filter result. ....	57
Figure 3.9: LPF code. ....	60
Figure 3.10: Get Sonar code. ....	60
Figure 3.11: The main code that gets the real data from Sonar1() and send it to LPF(). Eventually, it shows the result on Figure 3.12. ....	61
Figure 3.12: LPF results. ....	62

Figure 4.1: Kalman Filter Algorithm [12].....	64
Figure 4.2: Estimation process in LPF [12].....	68
Figure 4.3: $x_k \sim N(x_k, P_k)$ .....	69
Figure 4.4: Prediction process in Kalman Filter. Simply, the previous estimation $x_{k-1}$ is not used in the process of estimating the current state. Instead, the previous estimation is converted to a prediction $x_k$ – using the prediction equation: $x_k = Ax_{k-1}$ . Then the prediction of the current state is used to calculate the current estimation [12]. .....	71
Figure 4.5: State space equations diagram.....	74
Figure 4.6: A mass pushed by f. The outputs are p, v and a. ....	77
Figure 4.7: Simple RC circuit. ....	78
Figure 4.8: Robot following the right wall. The two sonars on the right detect the wall but the distance is not very accurate. In this case, Kalman filter is used to refine the measurements. ....	83
Figure 4.9: Simple1DKalman function (it gets the voltage and returns the estimate).....	85
Figure 4.10: GetSonar1 (A function to simulate the first sonar).....	85
Figure 4.11: GetSonar2 (A function to simulate the second sonar). ....	85
Figure 4.12: The main code. ....	87
Figure 4.13: Kalman Filter result.....	87
Figure 4.14: Kalman Filter algorithm code. This code gets the sonar data from the main code. Then, it computes the position and velocity and returns them to the main code to be shown.....	91
Figure 4.15: This code reads a file that contains a real sonar data. It sends the data to the main code. ....	92
Figure 4.16: The main code that gets the measurements from GetMaxSonar and passes them to the Kalman Filter algorithm code. After getting the estimated position and velocity, it will show the results. ....	93
Figure 4.17: Position estimation. ....	94
Figure 4.18: Velocity estimation. ....	94
Figure 4.19: Kalman Filter algorithm code. It receives the sonar data from the main code and sends the estimation back to the main code to be presented. ....	96
Figure 4.20: The velocity generator code. This code emulate the velocity of the robot... ..	96
Figure 4.21: The main code that receives the sensor data from GetVelocity() and passes it to Kalman Filter algorithm. Then, it show the result of Kalman Filter estimation. ....	97
Figure 4.22: Velocity estimation. ....	98
Figure 4.23: Position estimation. ....	98
Figure 4.24: $x_k \sim N(x_k, P_k)$ .....	99
Figure 4.25: Two dimension Gaussian.....	100
Figure 4.26: $1/100 Q$ . ....	101
Figure 4.27: $1/100 R$ . ....	102
Figure 4.28: The original axes are red and the new axes are gray after Yaw angle rotation [13].....	104
Figure 4.29: Sinusoidal signal used to test gyroscope and accelerometer. ....	105

Figure 4.30: Euler Angles. ....	106
Figure 4.31: Gyros Angular velocities. ....	107
Figure 4.32: GetGyro code. ....	107
Figure 4.33: Convert gyro measurements to Euler angles. ....	108
Figure 4.34: The main code. ....	109
Figure 4.35: Roll angle using gyro. ....	110
Figure 4.36: Pitch angle using gyro. ....	110
Figure 4.37: Accelerometer measurements simulator. ....	112
Figure 4.38: Convert Accelerometers' measurements to Euler angles. ....	113
Figure 4.39: The main code. ....	114
Figure 4.40: Angular velocities measured by Accelerometers. ....	114
Figure 4.41: Roll angle using the accelerometer. Equation 4.7 is used to convert the angular velocity to Euler angle. ....	115
Figure 4.42: Pitch angle using accelerometer. Equation 4.7 is used to convert the angular velocity to Euler angle. ....	115
Figure 4.43: Sensor fusion diagram. ....	116
Figure 4.44: Kalman Filter algorithm. ....	120
Figure 4.45: Convert Euler angles to Quaternion. ....	120
Figure 4.46: The main code that tests Kalman Filter algorithm. ....	121
Figure 4.47: Roll angle after using Kalman Filter sensor fusion technique. ....	123
Figure 4.48: angle after using Kalman Filter sensor fusion technique. ....	123
Figure 4.49: Extended Kalman Filter algorithm [12]. ....	124
Figure 4.50: Extended Kalman Filter function that receives arguments and return the estimation. ....	130
Figure 4.51: The code that test teh EKF and draw the results. ....	130
Figure 4.52: Roll angle estimation result of EKF. ....	131
Figure 4.53: Pitch angle estimation result of EKF. ....	131
Figure 5.1: MCECS Bot diagram [14]. ....	135
Figure 5.2: PEOPLE-Bot. ....	137
Figure 5.3: The first part of the base. ....	138
Figure 5.4: PSU Guide Robot (MCECS-Bot). ....	139
Figure 5.5: Torque calculation. ....	141
Figure 5.6: PSU Guide Robot [18]. ....	145
Figure 5.7: Square shape aluminum bar. ....	146
Figure 5.8: L shape aluminum bar. ....	146
Figure 5.9: Socket screw bolt. ....	147
Figure 5.10: Machine screw bolt. ....	147
Figure 5.11: U shape bolt. ....	148
Figure 5.12: U shape bolt is used to attach the DC motor and its gear to the main frame. ....	148
Figure 5.13: Nylon nut. ....	149
Figure 5.14: Different brackets that used in the main frame. ....	150
Figure 5.15: Flat washer. ....	150
Figure 5.16: Motor spacer and its holding washer. ....	151

Figure 5.17: 1/2" × 3/4" × 1/2" bronze spacer .....	151
Figure 5.18: Aluminum spacer .....	152
Figure 5.19: Omni Wheels alignment [20] .....	153
Figure 5.20: Mecanum wheels alignment [21] .....	154
Figure 5.21: Mecanum wheel movement profiles [23] .....	155
Figure 5.22: RoboClaw 2x15A [24] .....	156
Figure 5.23: DC motor [25] .....	157
Figure 5.24: Motor gearbox and the required pinion to attach the motor to the gear [16]. .....	158
Figure 5.25: Encoder [26] .....	158
Figure 5.26: gears to attach the encoder to the gear shaft .....	160
Figure 5.27: Gearbox 500 hub key .....	161
Figure 5.28: Gearbox key .....	161
Figure 5.29: Arduino Mega .....	162
Figure 5.30: LiFeMnPO4/60AH Battery pack [27] .....	164
Figure 5.31: LV - MaxSonar - EZ0 [29] .....	166
Figure 5.32: Sensors distribution on the robot .....	166
Figure 5.33: Sensors' holder [30] .....	167
Figure 5.34: Measurement taken from the front sonar. It was taken when the sonars are connected directly to the power without a signal to coordinate their work. It is obvious how noisy the signal is. Kalman Filter is used in the fourth chapter to refine this signal .....	168
Figure 5.35: Chaining method [31] .....	169
Figure 5.36: Delay circuit. The second group of sonars (six sonars) is connected to it. 170	170
Figure 5.37: The measurements of twelve sonars. It is obvious that the measurements are smooth and there is not noise except some sensors that show some noises. We were testing the sonars by moving a board in different directions and also test the angle that can be covered by each sonar. This test shows that the sonars are reliable and sensitive to any movements or object that comes in their range .....	170
Figure 5.38: Laser Range Finder [32] .....	171
Figure 5.39: Optical triangulation [33] .....	172
Figure 5.40: LRF mounted on server .....	172
Figure 5.41: PMC [35] .....	173
Figure 5.42: Four cells battery pack .....	174
Figure 5.43: Charger [36] .....	175
Figure 5.44: Power distribution Board .....	176
Figure 5.45: Micro switch .....	177
Figure 5.46: Limit switches that are used in the bumpers .....	178
Figure 5.47: These images show the steps of constructing the bumpers .....	179
Figure 5.48: Base dimension .....	180
Figure 5.49: The construction of Aluminum bars .....	181
Figure 5.50: Brackets holding two bars .....	181

Figure 5.51: Mecanum wheel is connected to the gearbox and DC motor and mounted on the aluminum bar.....	182
Figure 5.52: U shape bolt holds the gearbox to the main frame. ....	183
Figure 5.53: MCECS Bot's waist.....	184
Figure 5.54: Robot's Arm.....	184
Figure 5.55: MCECS-Bot's neck. ....	185
Figure 5.56: The head consists of an aluminum mask that covers the tablet which represents the head. On the right, a screen shoot from the tablet shows the face of the butler. ....	186
Figure 5.57: MCECS-Bot's software diagram. ....	187
Figure 5.58: Reactor Loop. ....	191
Figure 5.59: MRPT Libraries [37]. ....	194
Figure 5.60: MCECS-Bot circuit diagram. ....	196
Figure 7.1: Four sonars used in the wall following algorithm. ....	218
Figure 7.2: Wall following flow chart. ....	219
Figure 7.3: Main Kalman Filter code.....	222
Figure 7.4: The code that Test Kalman Filter algorithm. This code passes four arguments which are the measurements coming from the sonars used in the wall following. ....	224
Figure 7.5: The zero removal algorithm refines the zero measurements from Sonar 1 using Kalman filter. ....	226
Figure 7.6: The zero removal algorithm refines the zero measurements with Kalman filter. ....	226
Figure 7.7: The zero removal algorithm refines the zero measurements with Kalman filter. ....	227
Figure 7.8: The zero removal algorithm refines the zero measurements with Kalman filter. ....	227
Figure 7.9: The steps of Extended Kalman Filter [12]. ....	230
Figure 7.10: One dimensional Kalman Filter example. ....	231
Figure 7.11: The code that simulates the measurements of the encoders. ....	234
Figure 7.12: The code that simulates the ideal measurements of the encoders. ....	235
Figure 7.13: The code that simulates the measurements of sonar 12. ....	235
Figure 7.14: The code that simulates the measurements of sonar 12 but without noise or zero measurements. ....	236
Figure 7.15: Extended Kalman Filter code. ....	237
Figure 7.16: The main code of the one-dimensional EKF simulator. ....	238
Figure 7.17: One-dimensional EKF result. ....	239
Figure 7.18: Running the test of the EKF. ....	241
Figure 7.19: This window is produced by MRPT which is the map of the basement floor of the fab building. ....	242
Figure 7.20: This is the terminal window that runs the MRPT code. As shown in this figure, the information related to the robot position based on the sonars and encoders as well as the EKF estimates are presented in this terminal. The EKF estimate is zero here because the robot didn't start moving yet when this screen shoot	

were taken. The variable `front_sonar = 2.865` and the variable `initial front wall distance = 2.865` which means the robot is at the beginning of its path. ... 243

Figure 7.21: The biggest object, which represents the robot, is produced by EKF. It follows the ball that represents the robot position calculated based on the sonars' measurements. the two balls behind the robot are the initial position of the robot and the position of the robot based on the encoder's measurements. The window on the left reviews the information related to these objects..... 244

Figure 7.22: The first ball on the right (or the one that precedes the robot) is the robot's position based on the encoder's measurements. The ball that behind the first ball is the position calculated based on the sonars' measurements. It is obvious that the robot is affected by the sensors more than encoders. The EKF neglected the encoder accumulated error. The window on the left reviews the information related to this process. .... 245

Figure 7.23: The final position of the robot after it stopped moving. The robot position, which is based on the EKF estimates, is between the sonars' based position and the encoders' based position. It is closer to the Sonars because it has to eliminate the accumulated error of the encoders. The cone here is small because the robot is very close to the wall (the divider shown in Figure7.18). .... 246

Figure 7.24: The result of using 1D EKF. The sonars measurements were good. There was an accumulated error came from encoders. The estimates is more affected by sonars than by encoders. .... 247

Figure 7.25: The robot's different poses while moving from y-axis to x-axis. .... 249

Figure 7.26: The code that simulates the odometry measurements on x-axis. The measurement variable `b` (in the code) is passed by the main code. It represents the order of the measurement. It is used to generate different area of measurements. As seen later in EKF results, the encoder has some area when it increases. Additionally, this code generates an error that is added to the new measurements. This error represented by `w`. .... 253

Figure 7.27: The code that simulates the odometry measurements on y-axis. Variable `b` and `w` work in similar way as in `GetEncoder_X`. .... 254

Figure 7.28: The code that simulates the real position of the robot on x-axis. Variable `b` works in similar way as it does in `GetEncoder_X`. Variable `w` here equals zero as there is no error. .... 255

Figure 7.29: The code that simulates the real position of the robot on y-axis. Variable `b` works in similar way as it does in `GetEncoder_X`. `w` here equals zero as there is no error. .... 256

Figure 7.30: The code that simulates `sonar1` measurement which is the distance from the front wall. Variables `b` and `w` work in a similar way as in `GetEncoder_X`. 258

Figure 7.31: The code that simulates `sonar10` measurement which is the distance from the right wall. `b` and `w` work in similar way as in `GetEncoder_X`. .... 259

Figure 7.32: The code that simulates the measurements of the front sonar without noise or zero measurements. Therefore, `w` equals zero here. .... 260

Figure 7.33: The code that simulates the measurements of the right sonar without noise or zero measurements. Therefore, $w$ equals zero here. ....	261
Figure 7.34: Two-Dimensional Extended Kalman Filter code. ....	262
Figure 7.35: The main code that coordinates the process of simulating 2D EKF.....	264
Figure 7.36: The result of the 2D EKF.....	265
Figure 7.37: The robot is now stationary. Probably, it is taking measurements using sonars and initializing the front wall and the right wall distances. This happens when the robot receives a command to move. The sonars cone is wider here because the sonars used here are different than the sonars used with 1D EKF. The sonars used in the 1D EKF were sonar1, sonar2, sonar11 and sonar12. In 2D EKF, the sonars used are sonar1, sonar3, sonar10 and sonar12. The reason is because in 2D EKF the robot is detecting walls or objects in different directions which is not only detecting the objects that are in the front of the robot as assumed in 1D EKF.....	267
Figure 7.38: The first step of the robot is shown in the current figure. This figure shows that the position based on the sonars was updated but the one based on the encoders was not updated yet. However, the robot followed the sonars. ..	268
Figure 7.39: New object is shown here, which is the position of the robot based on the encoders' measurements. ....	269
Figure 7.40: The robot's position based on sonars' measurements is updated here. The robot is closer to sonars than to encoders. This is happened because the robot is affected by measurements (sonars) more than by the old estimate. This can be modified if the results is not satisfactory by changing the value of matrices $R$ and/or $Q$ . ....	270
Figure 7.41: The robot start to move right here. Both position based on encoders' and sonars' were updated here. ....	271
Figure 7.42: The last position of the robot. The estimate was closer to sonars' measurements. In other words, it didn't affected by the accumulated error of the encoders. ....	272
Figure 7.43: As shown, the estimate was little bit affected by the wrong measurements of the sonars. However, it wasn't affected for long time. The final estimation of the robot's position on x-axis is smaller than the one based on right sonar's measurements.....	273
Figure 8.1: The structure of Mobile Robot Localization based on Kalman Filter thesis.	276
Figure 8.2: Different movement profiles of Mecanum wheels [23]. ....	280
Figure 8.3: This figure shows a Map of a building and all the equations related to the robot's poses which are used later to derive equations' model of the EKF. ....	282
Figure 8.4: Data accessed on the Samsung Galaxy tablet. ....	283
Figure 8.5: Wi-Fi signal strength representation.....	284

## Mobile Robot Localization Based on Kalman Filter



Chapter 1: Introduction

## **Chapter 1: Introduction**

Robotics is one of the fastest growing areas of research. It will contribute much to the future and has many promising properties. Nowadays, robotics is everywhere and is affecting our lives tremendously. Starting from computers and ending with space crafts. Humanity is getting more dependent on robots and every year there are thousands of researches and projects that add new contributions to science and engineering of robotics. New algorithms that improve old techniques as well as new designs and new robots are presented continuously. The ultimate goal is to develop a fully automated machine with all its required sensors to sense its surroundings and make behavioral decision independently. Additionally, the machine that would work in a social environment should have also a reasonable size and behaviors based on perceived personality. Autonomous behavior is one of the most important aspects related to modern robotics. Localization and navigation are part of automatic behavior and they are one of the most quickly developing fields of robotics research presently.

Localization is a process that occurs inside the robot “brain” to locate the robot in a previously known map. In other words, the map is already located in the robot memory and is based on landmarks of the environment that the robot perceives. Robot uses sonar, camera, Laser Range Finder (LRF) and stereo-vision devices like Kinect to perceive the environment. Using localization and sensors, robot will be able to detect its position like  $x$ ,  $y$  coordinates as well as its orientation in the map. Even the sensing for localization process is not as easy as presented above because there are many types of sensors and a sensor fusion are required in the process. As long as there are sensors, there are noise and

errors which cause uncertainty about all data produced by sensors. Always, without a refining algorithm, the outcome of the process of locating the robot position will diverge. In this case, if we need our robot to be in the hall room, after a while it will find itself in the kitchen! Nevertheless, filters are used by the robot in order to figure out the best estimation of its location and position. Methods of using a filter to refine signals coming from the sensors in order to determine the robot's position will be the main topic of this thesis.

To apply all different theories and algorithms as well as inventing new ones, a suitable robot should exist in the first place. In Fall 2011, Professor Perkowski was asked by Dean Prof. Su to build a guide robot that will give guided tours to the Engineering building. The first robot that served for this purpose was the PEOPLE-Bot (Figure 1.1). Jim Larson was the leader of the team that worked on the PEOPLE-Bot in Fall 2011. They fixed a lot of technical issues and improved the board circuits of this robot. A new features were added to this robot, such as navigation, speech recognition and gesture recognition. Due to the limited functionality and ability of the PEOPLE-Bot, professor Perkowski decided to have a better and more functional robot. Many meetings were set with him in order to decide what kind of base the task required, what the shape for our new robot was going to be as well as what the possible functions to be handled by the robot were.



Figure 1.1: PEOPLE-Bot.

The new robot name is MCECS-Bot (Maseeh College of Engineering and Computer Science Bot) which is shown in Figure 1.2. MCECS-Bot is “an intelligent autonomous Guide robot that will give guided tours and be able to lead a user to a sequence of specific locations in the basement area of the Engineering Building and the Fourth Avenue Building at Portland State University.” [1]. I was chosen by Professor Perkowski to be the leader of MCECS-Bot team. I made the robot design, especially the base. After couple meeting with Professor Perkowski, the decision was made to choose the Mecanum wheels to be the wheels for the base. The design calculations were made by me to in order to choose suitable parts for the robot. The main design consideration was the torque calculation which was made in order to specify the supported weight, motors, gears, speed,

power consumption and selection of the main controlling boards. The main physical structure took me two terms to finish. During these terms, Winter and Spring 2012, other teams were working on the arm, neck waist as well as improving the other software that existed previously on the PEOPLE-Bot. Basically, PEOPLE-Bot was created as a universal platform to develop and test main algorithms as well as all support software that will be used on MCECS-Bot.

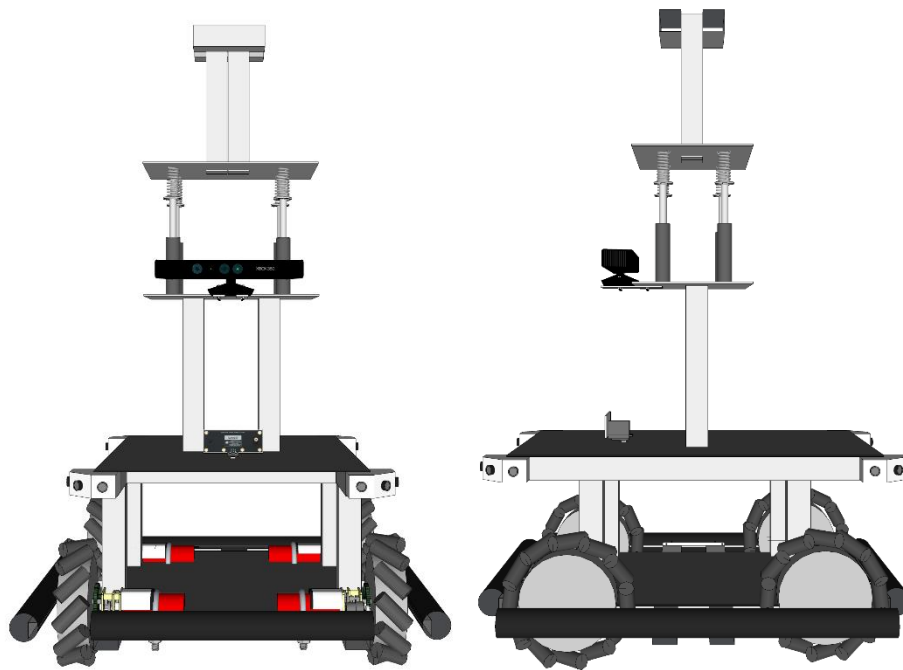


Figure 1.2: MCECS-Bot (PSU Guide Robot) [2].

I built the main part of the robot which is the base and all its required mechanical and circuit's boards. The main structure is made using aluminum bars. The mechanical part of the base consists of the main structure that contains the circuit boards and hardware parts like wheels, motors, gears, encoders, motor controllers, Bumper circuits, batteries and the

Power Management Circuits (PMC). In the Fall 2012, when another new team was involved in this project, they added new sensors like LRF and Kinect. A Power distribution board was added to distribute the power from battery and connect to the charger. Their team implemented some navigation processing software for the robot. I contributed to their work in the mechanical design as well as software. I worked on the power distribution board and fixed couple issues as well as updated the board layout to get different voltages. Chapter five will be dedicated for the entire mechanical and system design of MCECS-Bot.

Navigation is the next step taken by the robot when it knows its position. Localization is the main part here as the robot must know its position precisely while driving from point to point. The navigation scenario will be:

1. The robot knows its essential position.
2. Robot receives an order to go to a different position (or set this goal by himself).
3. Robot creates a trajectory step by step from its start point to its destination point. This trajectory is first specified as a path in the map and then it is converted to control commands for the robot.
4. Starts moving.
5. Keep localizing itself while moving as it has to calibrate itself every while.

Here, more sensors are involved in navigation and localization (for example; odometry will be considered after the robot starts moving).

It is known that finding the closest estimation to the real values is very important in localization process. There are many ways of finding the best estimation of the real values. The robot uses its sensors to get uncertain measurements and the easiest method that can be used is to refine these measurements by averaging the incoming values.

Based on our experience so far as well as literature we decided to add another method for a more advanced robot, based on filters such as Kalman Filter and Particle Filter. Although the filter methods are more complicated, they are nowadays a standard in the most advanced robots such as military “search and rescue” robots and industrial robots and we decided to have various methods implemented on the robot in order to compare their operation. Based on study of literature and previous work done by our team, I decided to select the Kalman Filter to be considered as a controller for MCECS-Bot because other students work on equivalent methods such as Particle Filter. Kalman Filter is used to filter all the measurements and to find the closest or the best results that converge the real values. Kalman Filter is an interesting prediction/filtering algorithm that gives very precise estimations. For example; if the robot measures the position over time which is accompanied with an error, Kalman Filter is able to give the best estimation of the robot’s position. It will also give the velocity which the robot didn’t even measure (it is explained later in this thesis by some examples in chapter 3).

In order to study Kalman Filter, an introduction to its algorithm is required. An extensive introduction to Kalman Filter is presented in this thesis. The presentation of Kalman Filtering ideas will be divided into two chapters. The first chapter is only a mathematical introduction because Kalman Filter is an advanced area of study based on

many areas of mathematics that are not normally studied at the M.S level studies in engineering. The explanation will start with basic concepts of probability theory, normal distribution, matrices and other basic mathematical concepts required to understand Kalman Filter in full detail and next propose some concrete improvements to it in our setting. The next chapter is about other simple filters that will lead ultimately to the explanation of the Kalman Filter which is too complicated to explain directly. Recursion is one of the main Kalman Filter's features. It will be explained by comparing it with other similar filters. Moreover, both mathematical introduction and filters' recursive theory introduction will be explained on many examples in order to make the material covered by this thesis easy to understand by future students who will continue working on this or similar projects.

The reason of developing this thesis is to apply Kalman Filter to control the MCECS-Bot (or PSU guide robot). MCECS-Bot is made of different parts; base, waist, arms, neck and head. It is a humanoid robot that has a face to use facial gestures for interaction with visitors. MCECS-Bot has four Mecanum wheels which allow the base to have three degrees of freedom, waist that consist of four linear actuators that allows the robot to bend forward/backward and left/right, two arms, neck and head. There are many sensors that provide the robot with the sense of the surrounding environment. My thesis has two main components; mechanical design and electrical design of the robot base and the main body without hand and head, and software for navigation and localization together with its integration with a complete autonomous system of the robot. The procedures of building this robot from scratch and how this goal has been achieved are covered by this



thesis as well. To put it simple, I am responsible for all motion problems of the robot base, while other members of the team are responsible for human interaction and guiding functions.

The ultimate goal of this thesis is to have a comprehensive presentation of both the mathematics related to Kalman Filter and application of Kalman Filter for this particular robot base design with Mecanum wheels and relatively large size of a humanoid robot, a topic rarely covered in the literature. Second, some other simple filters also presented to explain how the recursive process works which is another introduction to Kalman Filter. Third, Kalman Filter and Extended Kalman Filter are explained together with simple examples illustrating their behavior in practical control and prediction situations. Fourth, MCECS-Bot design and description are given. Finally, the main chapter of this thesis presents implementation of Kalman Filter on the specific robot mechanical/electrical architecture to the Mecanum base holonomic control of MCECS-Bot.

Chapter 2: Mathematical Introduction

## **Chapter 2: Mathematical Introduction:**

This chapter is as an introduction to Kalman Filter. In order to be familiar with Kalman Filter and the mathematics on which it is based, a brief introduction to different fundamental mathematical operations involved will be presented. Every subject will be explained with many examples to make it easy to understand. The rule of thumb is to have more simple examples that explain various ideas in order to get the reader ultimately involved with the fundamental topics of this material.

### **2.1 Permutation**

Permutation is an arrangement of all possible collection or selecting of some objects (elements of a set) where the order does matter. Permutations can be:

#### **2.1.1 Permutation with Repetition**

The role of finding the permutation size (or how many permutations) is:

Number of permutations with repetition =  $n^r$  (n: number of things that we're going to choose from and r is the size of our choice).

##### **2.1.1.1 Example**

We have combination lock with 10 numbers (0-9) and we have to choose 3 numbers only. Each time we have 10 choices, so the total number of possible opening sequences will be  $10 \times 10 \times 10 = 10^3 = 1000$ .

## 2.1.2 Permutation without Repetition

Permutation without repetition means if there is a number that was chosen first time, we can't choose it again. To know how many permutations without repetition exists, the formula below is used:

$$\text{Number of permutations without repetition} = n!$$

### 2.1.2.1 Example

We have nine balls with different colors and we want to arrange them. When we choose a ball we can't choose it again. The Permutation of nine balls is  $9! = 362,880$ . This number tells us how many different arrangements the nine balls have. If we want to arrange a certain number which means choosing numbers to arrange out of a bigger population of numbers, the below equation is used:

$$\text{Permutation without repetition} = n! / (n-r)!$$

(n: number of things that we're going to choose from and r is the size of our choice).

### 2.1.3 Example

To choose/arrange 3 balls out of nine balls of the previous example:

$$9! / (9-3)! = 504.$$

## 2.2 Combination

Combination is a way of selecting several things out of a larger group (order does not matter). Combination can be in two categories:

## 2.2.1 Combinations without Repetition

### 2.2.1.1 Example

We have three balls (Red, Blue and Green)

The set of permutations is (RBG, RGB, BGR, BRG, GRB, and GBR). The number of permutations is 3! It is obvious that the repetition matters here.

The combination of these balls is only (RGB) as the order does not matter.

The rule of finding combination without repetition:

$$n! / r! (n - r)! = \binom{n}{r}$$

Where n is the number of things that we're going to choose from and r is the size of our choice.

### 2.2.1.2 Example

Back to our nine colored balls, if we want to select 4 out of 9

$$= 9! / 4! (9 - 4)! = 126$$

## 2.3 Standard Deviation

Standard deviation is a way to tell us how the data are spread out. Standard deviation is also a square root of variance. Standard deviation is important as it tells us whether the data are close to the mean or not. It is the square root of the average weighted

of the difference between each sample and the mean value. The equation that calculates Standard deviation is:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

We need to calculate the variance in order to find the standard deviation. In this case we need to know what the variance is. Variance is an indication about how the data is spread out around the mean. The bigger the number is the more spread out the data gets. On the other hand, the smaller the number is the closer the data gets. The equation for variance is:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$$

### 2.3.1 Variance and Standard Deviation Example [2]

Easy example to illustrate the importance of the standard deviation. In this example, there are five dogs with different heights as shown in the picture below (Figure 2.1).

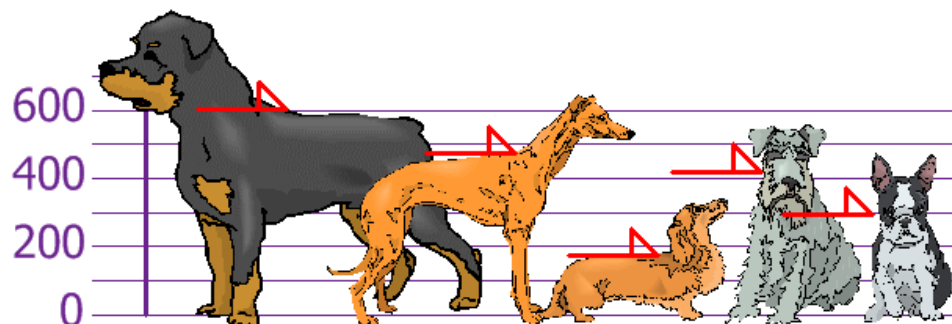


Figure 2.1: Standard deviation example.

First we need to find the mean value:

$$\text{Mean} = \frac{600 + 470 + 170 + 430 + 300}{5} = 394$$

394 is just a number that tells us the arithmetic average of these dogs' heights. However, it doesn't tell us how the heights distributed whether they are close to the mean or not. We can get the same value (394) with different heights than the ones given in the previous example.

$$\text{Variance: } \sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2 = \frac{206^2 + 76^2 + (-224)^2 + 36^2 + (-94)^2}{5} = 21,704$$

$$\text{Standard deviation} = \sqrt{21,704} = 147$$

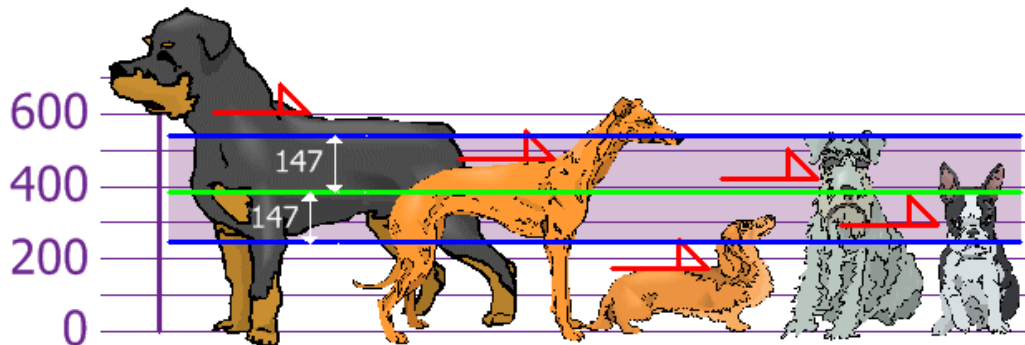


Figure 2.2: Standard deviation example.

In Figure 2.2 the value 147 (with mean which is the line in the middle) tells us about the dogs that have a normal height. In other words, which one is shorter than normal and which one is higher than normal.

## 2.4 Expected Value

To calculate the mean (expected value) we perform the summation of each outcome value multiplied by the number of its occurrences. This gives a sum S that is next divided by the total number of objects. It is the weighted average of all possible values that a specific random variable can take.

### 2.4.1 Example

Let's start with simple example, suppose we have these random values: 2, 2, 5, 5, 5, 7, 9 and their mean is:

$$\text{Mean} = (2+2+5+5+5+7+9)/7 = 35/7 = 5$$

Observe that expected value is also the total number of averages 2, 5, 7 and 9, multiplied by their frequencies  $2/7$ ,  $3/7$ ,  $1/7$  and  $1/7$ , respectively, as follows:

$$\text{Or: } \frac{2(2)+3(5)+7+9}{7} = 2/7 \cdot (2) + 3/7 \cdot (5) + 1/7 \cdot (7) + 1/7 \cdot (9)$$

$$= 28.5\% \cdot 2 + 42.8\% \cdot 3 + 14.2\% \cdot 7 + 14.2\% \cdot 9 = 5$$

The percentage is how likely we will get each specific value, in other words the probability that each element will show up.

Some notes about the Expected value:

1. Expected value is the same as a population mean.
2. The expected value doesn't have to be the most probable value.



3. We use expected value instead of mean because we can't add up all the values in the entire population (what about infinite numbers) and divide by the size of that population.

### **2.4.2 Example**

Another common example of calculating the expected value is the expected value of the outcome of a six-sided dice.

The probability of getting each side is  $1/6$ . Therefore:

$$E(X) = 1 \cdot 1/6 + 2 \cdot 1/6 + 3 \cdot 1/6 + 4 \cdot 1/6 + 5 \cdot 1/6 + 6 \cdot 1/6 = 3.15$$

There is a general rule which says that if each outcome has the same probability, then, the expectation calculation becomes simply a weighted average as shown in the example above.

## 2.5 Probability

A measure of how likely it is that a certain thing will happen. Probability has a value between 0 and 1. The picture below (Figure 2.3 [3]), is a simple illustration of probability.

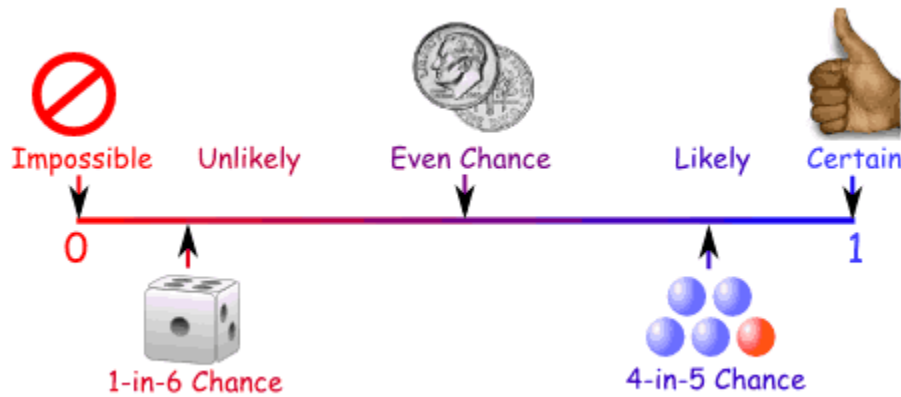


Figure 2.3: Probability illustration.

### 2.5.1 General Rules

- $P(A) = \frac{\text{Possible outcomes favoring event A}}{\text{Total number of possible outcomes}}$
- The probability of an outcome favoring either A or B is given by:  
$$P(A \cup B) = P(A) + P(B) - P(A) \cdot P(B)$$
- The probability of two independent outcomes is:  
$$P(A \cap B) = P(A) \cdot P(B)$$
- Conditional probability: the probability of the outcome given an occurrence of outcome. B:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A \cap B) = P(A|B) \cdot P(B), \quad P(B \cap A) = P(B|A) \cdot P(A)$$

$$P(A|B) \cdot P(B) = P(B|A) \cdot P(A)$$

$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)} = \frac{P(A \cap B)}{P(A)}$$

### 2.5.2 Simple Examples of using Probability

Upcoming, are many examples that will cover the probability concept in order to make Probability fully understandable simply and straightforwardly.

- If we have a six sided dice, we can find the probability of having even or odd numbers,  $P(1, 2, 3, 4, 5 \text{ or } 6)$  or we can find the  $P(<5)$  (the probability of having a number less than five), etc.

$P(\text{even}) = \frac{1}{2}$ , the dice has 6 sides and each side has a number. In total there are a set of numbers from 0 to 6. Half of these numbers are even and the other half is odd. The probability of even is telling how likely we will get an even number if we through the dice once.

$$P(\text{odd}) = \frac{1}{2}.$$

$P(\leq 5) = \frac{5}{6}$  which is the probability of having a number that is not larger than 5.

- Tossing coin probability:

$$P(\text{TH} \cup \text{HT}) = P(\text{HT}) + P(\text{TH})$$

$$= 1/4 + 1/4 = 1/2. \text{ TH means the}$$

probability of having tail then head.

HT is the probability of having

head then tail. Figure 2.4 illustrates the

probability of having a consecutive heads or tails after tossing a coin for two times.

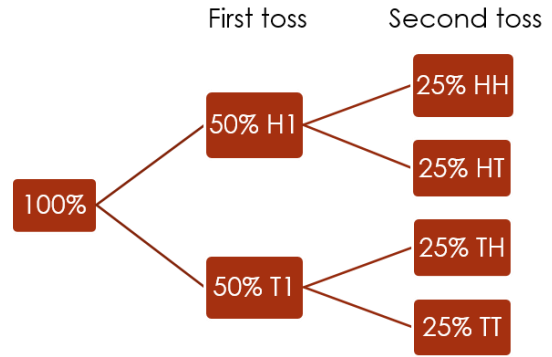


Figure 2.4: Probability diagram.

- $P(5 \text{ consecutive heads}) = 1/2 \cdot 1/2 \cdot 1/2 \cdot 1/2 \cdot 1/2 = 1/32$
- $P(\text{getting no heads after 6 consecutive tosses}) =$   
 $P(6 \text{ consecutive Tails}) = (1/2)^6 = 1/64$
- $P(\text{HHHHHT}) = 1/64$  too!
- $P(\text{Exactly 1 head only out of 5 tosses}) = 5/32$
- $P(\text{Not getting exactly 1 head}) = 1 - 5/32 = 27/32$
- $P(\text{At least one head out of 5-tosses}) = 1 - P(\text{No head out of all tosses - all tails})$   
 $= 1 - 1/32 = 31/32$
- I have 9 normal coins in my pocket and one two-heads-sided coin [4]. The probability of getting 5 consecutive heads if I pick one coin and flip it 5 times is:  
 The probability of getting a normal coin which equals the probability of getting five consecutive

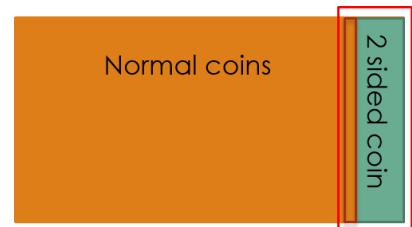


Figure 2.5: The distribution of coins.

heads from normal coin added to the probability of getting a five consecutive heads from the two-sided-coin.

$$(1/10 \cdot 1) + (9/10 \cdot 1/32) = 9/320 + 32/320 = 41/320. \text{ The detailed solution}$$

for each case related to this example is listed below:

- The whole rectangle (Figure 2.5) represents the total number of coins which is 10.
- (Color oriented) Number of normal coins = 9
- No. of two-sided coins = 1
- The probability of having five consecutive heads and the coin is normal:

$$P(5/5|N) \cdot P(N) = 1/32 \cdot 9/10 = 9/320 = P(5/5 \cap N)$$

- The probability of having five consecutive heads and the coin is a two sided tail:

$$P(5/5 \cap 2s) = P(5/5|2s) \cdot P(2s) = 1 \cdot 1/10 = 1/10$$

- The probability of having five consecutive heads:  $9/320 + 1/10 = 41/320 = P(5/5)$
- The probability of having normal coin if the result is five consecutive heads after

$$\text{flipping the coin five times: } P(N|5/5)^* = \frac{P(N \cap \frac{5}{5})}{P(\frac{5}{5})} = \frac{\frac{9}{10} \cdot \frac{1}{32}}{\frac{41}{320}} = \frac{9}{41}$$

- The probability of having two-sided-coin coin if the result is five consecutive heads

$$\text{after flipping the coin five times: } P(2s|5/5) = \frac{P(2s \cap \frac{5}{5})}{P(\frac{5}{5})} = \frac{\frac{1}{10} \cdot 1}{\frac{41}{320}} = \frac{32}{41}$$

- A bag filled with 5 fair coins and 10 unfair coins with 80% H. Find the probability after 6 tosses  $P(\text{Fair}|4/6)$  (4 Heads out of 6 tosses). Detailed solution for different cases are explained below:

○  $P(4/6|Fair) = (\text{If I have fair coins, what is the probability of getting exactly 4 heads after 6 tosses?}) = \binom{6}{4} \cdot \left(\frac{1}{2}\right)^6 = 15/64.$

○  $P(6/4|Unfair) = (\text{If I have unfair coins, what is the probability of getting exactly 4 heads after 6 tosses?}) \binom{6}{4} \cdot (0.8)^4 \cdot (0.2)^2 = 24.57\%$

○ The probability of having 4 heads after toss the coin six times:

$P(\text{four heads/six tosses}) = (\text{probability of having fair coins} \cdot \text{probability of having four heads out of six tosses in the case that coins are fair}) + (\text{probability of having unfair coins} \cdot \text{probability of having four heads out of six tosses in the case that coins are unfair}).$

$$P(4/6) = 5/15 \cdot 15/64 + 0.2457 \cdot 10/15 = 24.19\%$$

○ The probability that the coin is fair if the result is four heads after tossing the coin six times:

$$P(Fair|4/6) = \frac{P(Fair \cap \binom{4}{6})}{P(\binom{4}{6})} = \frac{P(\binom{4}{6}|Fair) \cdot P(Fair)}{P(\binom{4}{6})} = \frac{5/64}{0.2419} = 32.29\% \text{ the}$$

probability that the current 4 heads out of 6 tosses came from Fair coins.

● We have 40 students in a classroom. What is the probability that there are two student share the same the same birthday?

○ It is easier to find the probability of people having distinct birthdays.

○ For two people:  $365/365 \cdot 364/365 = 365 \cdot 364/365^2$

○ For three people:  $365 \cdot 364 \cdot 363/365^3 = \frac{365!}{365^2}$

- For 40 people:  $P(d) = \frac{365 \dots 335}{365^{40}} = \frac{\frac{365!}{(365-40)!}}{365^{40}} = \frac{\frac{365!}{325!}}{365^{40}} = 10.87\%$
- $P(\text{Two student have the same birthday out of 40 people}) = 1 - P(d) = 100\% - 10.87\% = 89.13\%$  that somebody share the same birthday with someone else.

### 2.5.3 Binomial Probability (Discrete)

Binomial Probability is the way of representing a range of probabilities of an experience. However, each probability is exactly descriptive to the outcome. In other words, the outcome is either yes or no (or the outcome 0 or 1). This discrete probability tells us the range of successive experiments' outcomes. The main equation of calculating the Binomial distribution is:

$$P(x) = \binom{n}{k} P^k (1 - P)^{1-k}$$

#### 2.5.3.1 Example

Let's have an example of flipping a coin for six times again.

- Here the  $P(x) = \overline{P(x)}$
- The range of the outcomes of the experiment (Figure 2.6):
  - $P(0 \text{ head}) = \binom{6}{0} \cdot (0.5)^6 = 1/64$
  - $P(1 \text{ head}) = \binom{6}{1} \cdot (0.5)^6 = 6/64$
  - $P(2 \text{ heads}) = \binom{6}{2} \cdot (0.5)^6 = 15/64$

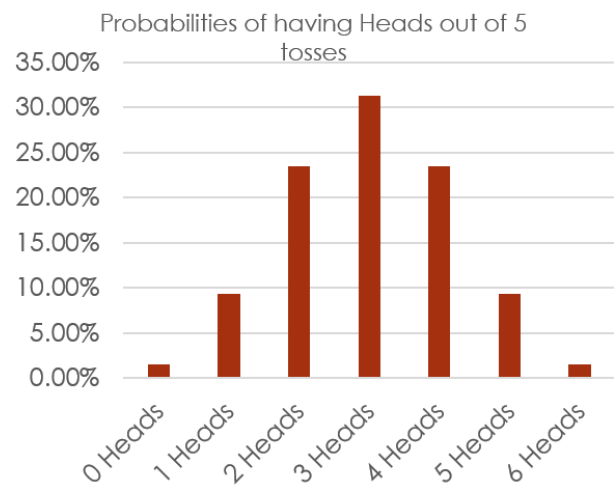


Figure 2.6: Binomial Distribution.

- $P(3 \text{ heads}) = \binom{6}{3} \cdot (0.5)^6 = 20/64$
- $P(4 \text{ heads}) = \binom{6}{4} \cdot (0.5)^6 = 15/64$
- $P(5 \text{ heads}) = \binom{6}{5} \cdot (0.5)^6 = 6/64$
- $P(6 \text{ heads}) = \binom{6}{6} \cdot (0.5)^6 = 1/64$

### 2.5.3.2 Example

Let's have another Example with different probabilities. What is the probability of having a rain in  $k$  days during a week if the probability of raining of each day is 80%? Figure 2.7 below show the binomial distribution of range of raining days during a week. The figure shows us what the probability of having rain in seven consecutive days is, if the raining probability is 80%.

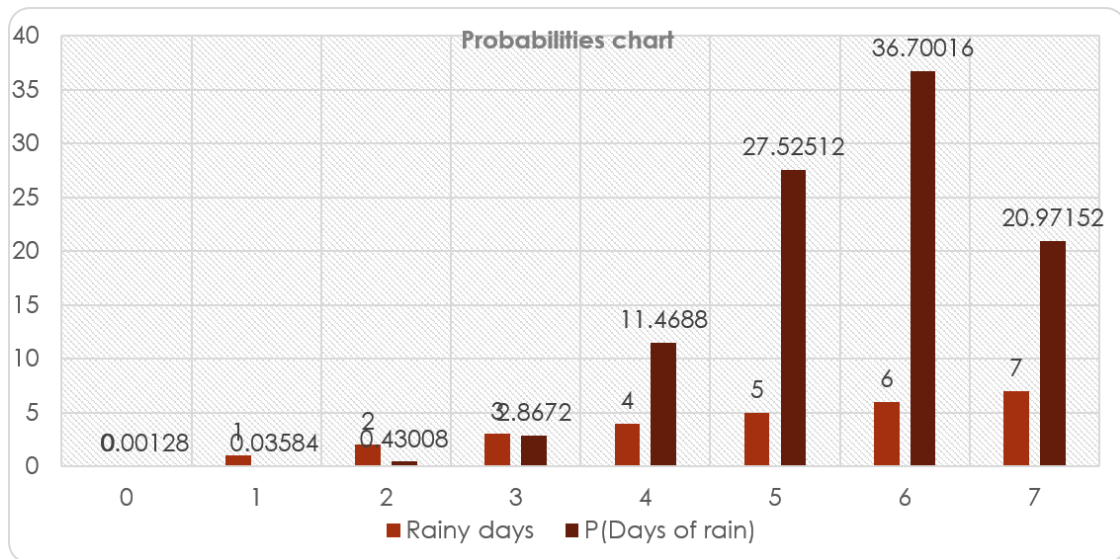
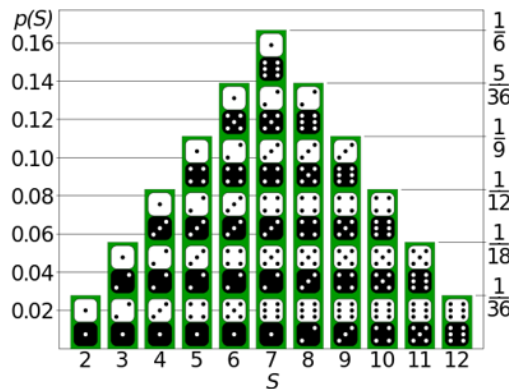


Figure 2.7: Raining probability.

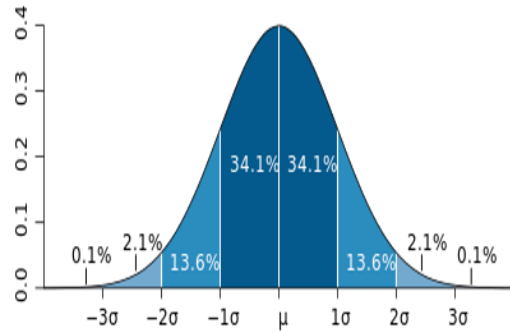


## 2.5.4 Probability Distribution Function

The probability distribution shows what the probability of each possible outcome of a random experiment is. The probability distribution can be for a discrete or continuous and for a non-cumulative or cumulative random values. Figure 2.8 shows two different examples of a random result of two different experiments (discrete and continuous).



Discrete probability distribution for the sum of two dice.



Normal distribution, also called Gaussian or "bell curve", the most important continuous random distribution.

Figure 2.8: Discrete and continuous probability distribution function examples [17].

### 2.5.4.1 Example

$X$  = continuous random value and we want to explore its probability distribution (Figure 2.9).

$$P(2 \leq X \leq 3) = 1 \cdot 1/5 = 1/5$$

$$P(3.9 \leq X \leq 4.1) = 1/5 \cdot 1/5 = 1/25$$

$$P(2.999 \leq X \leq 3.001) = 1/500 \cdot 1/5 = 1/2500$$

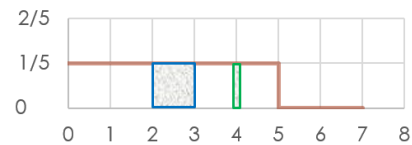


Figure 2.9: Probability distribution function example.

$$P(X = \text{exact } 3) = 0.$$

## 2.5.5 Expected Value of Binomial Distribution

In this section, a derivation of the expected value of binomial distribution is presented.  $X$  = Number of successes with probability  $P$  after  $n$  trials. Let's assume that I calculated the probability distribution for this example and we got a shape similar to bell curve. We need to prove  $E(X) = n \cdot P$ .

### 2.5.5.1 Example

$X$  = Number of baskets I make after  $n=10$  shots if I have a probability of making one shot is:  $P = 40\%$ .

$$E(X) = P \cdot n = 0.4 \cdot 10 = 4.$$

$$P(X=k) = \binom{n}{k} P^k (1 - P)^{n-k}$$

$$E(X) = \sum_{k=0}^n k \binom{n}{k} P^k (1 - P)^{n-k}$$

$$= \sum_{k=1}^n k \binom{n}{k} P^k (1 - P)^{n-k}, \text{ without } k \text{ the summation} = 1$$

$$= \sum_{k=1}^n k \cdot \frac{n!}{k! (n - k)!} \cdot P^k (1 - P)^{n-k}$$

$$= \sum_{k=1}^n \frac{n!}{(k - 1)! (n - k)!} \cdot P^k (1 - P)^{n-k}$$

$$= n \cdot P \cdot \sum_{k=1}^n \frac{(n-1)!}{(k-1)!(n-k)!} \cdot P^{k-1} \cdot (1-P)^{n-k}, \quad a = k-1, \quad b = n-1, \quad n-k = b-a$$

$$= n \cdot P \cdot \sum_{a=0}^b \frac{b!}{a!(b-a)!} \cdot P^a (1-P)^{b-a}$$

$E(X) = n \cdot P$ , this is true for only binomial distribution.

## 2.5.6 Conditional Probability

Dependent probability is a description of a case when the possible outcome of an event depends on the outcome of another event.

### 2.5.6.1 Example

A Bag is filled with 5 marbles (Figure 2.10). Pay \$0.35 to play and if you get 2 white in the first two attempts, you win \$1.

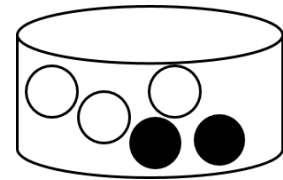


Figure 2.10: Bag filled with 3 white and 2 black marbles.

$$P(1^{\text{st}} \text{ white}) = 3/5$$

$P(\text{two white}) = P(1^{\text{st}} \text{ g} \cap 2^{\text{nd}} \text{ g}) \neq 3/5 \cdot 3/5$  as after choosing the first white marble the remaining are 4.

$P(1^{\text{st}} \text{ g} \cap 2^{\text{nd}} \text{ w}) = P(1^{\text{st}} \text{ w}) \cdot P(2^{\text{nd}} \text{ w} | 1^{\text{st}} \text{ w}) = 3/5 \cdot 2/4 = 6/20 = 3/10 = 30\%$  in average, pay \$0.35 and win \$0.3! (You play 100 times paying \$35 and the probability is 30% which means you lose \$5.)

What about playing with putting the white marble back? The probability now becomes independent. The result is:

$P(\text{two white}) = P(1^{\text{st}} w \cap 2^{\text{nd}} w) = 3/5 \cdot 3/5 = 9/25 = 36\%$ , in general you pay \$0.35 to get \$0.36, which means you win \$0.01.

### 2.5.7 Bayes' Theorem

Bayes' Theorem, or the related likelihood ratio, is the key to almost any procedure for extracting information from data. Bayes' Theorem lets us work backward from measured results to deduce what might have caused those [5].

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P((B \cap A) \cup (B \cap \bar{A}))} = \frac{P(B|A) \cdot P(A)}{[P(B|A) \cdot P(A)] + [P(B|\bar{A}) \cdot P(\bar{A})]}$$

#### 2.5.7.1 Example

Two students want to run an experiment. 1<sup>st</sup> student has a bag filled with stars (white and black). 2<sup>nd</sup> student has a bag filled with marbles (white and black). Each throw 5 pieces in a jar as shown in Figure 2.11:

- 1<sup>st</sup> student throws 4 white Stars and 1 black star

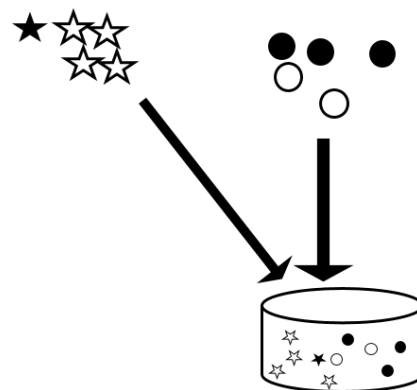


Figure 2.11: Jar filled with stars and marbles.

- 2<sup>nd</sup> student throws 2 white marbles and 3 black Marbles

Someone came and picked a piece from the Jar. He told us that he picked a black piece.

What is the probability that this piece is marble,  $P(M|B)$ ? Or  $P(S|B)$ .

$$P(M|B) = \frac{P(M \cap B)}{P(B)} = \frac{P(B|M) \cdot P(M)}{P(B)}, \quad P(B|M) = \frac{3}{5} = 60\%, \quad P(M) = \frac{1}{2} = 50\%.$$

$$P(B) = P((B \cap M) \cup (B \cap S)) = [P(B|M) \cdot P(M)] + [P(B|S) \cdot P(S)]$$

$= \left[ \frac{3}{5} \cdot \frac{1}{2} \right] + \left[ \frac{1}{5} \cdot \frac{1}{2} \right] = \frac{3}{10} + \frac{1}{10} = \frac{4}{10} = 40\%$  is the probability of getting a black object if someone picked an object from the jar.

$$P(M|B) = \frac{\frac{3}{5} \cdot \frac{1}{2}}{\frac{4}{10}} = \frac{\frac{3}{10}}{\frac{4}{10}} = \frac{3}{4} = 75\%$$
 is the probability of having a marble if someone

picked a black object from the jar.

## 2.6 Variance

Variance is a way to measure or see whether the data are close or far from mean.

Variance can be calculated for population or for a sample taken from a population if the population is too big, for example finding the variance of the men height in the U.S.

Variance is calculated using the formula below:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2 \quad \text{OR} \quad \sum_{i=1}^N (P_i \cdot (x_i - \mu)^2), \quad \mu = \sum_{i=1}^N P_i \cdot x_i$$

An easy example shows why we need the concept of variance. Let's take two groups of numbers:  $x = \{1, 2, 3, 4\}$ ,  $y = \{0, 0, 5, 5\}$ . The mean value for both groups is 2.5. We need something else to give some information about data (are they close to each other or to the mean?). We need a measure of dispersion.

$$\sigma^2(x) = \frac{(1-2.5)^2 + (2-2.5)^2 + (3-2.5)^2 + (4-2.5)^2}{4} = \frac{2.25 + 0.25 + 0.25 + 2.25}{4} = 1.25$$

$$\sigma^2(y) = \frac{(0-2.5)^2 + (0-2.5)^2 + (5-2.5)^2 + (5-2.5)^2}{4} = \frac{6.25 + 6.25 + 6.25 + 6.25}{4} = 6.25$$

$\sigma^2(y) > \sigma^2(x)$ , this result shows that the data in y group are far from the mean.

If the population is too big we need to calculate the variance for a sample using the formula below:

$$S^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{X})^2$$

The example in the Figure below shows the difference between variance for a population and a sample of a population. In the example below, the data are represented using black dots and there are two different samples; circles filled with crossed lines and black dots. The formula above is correct if the sample data are distributed (the circles that are filled with crossed lines) in Figure 2.12. If the data are clustered around some point in the space. The sample main ( $\bar{X}_1$ ) will be skewed from the population main ( $\mu$ ).

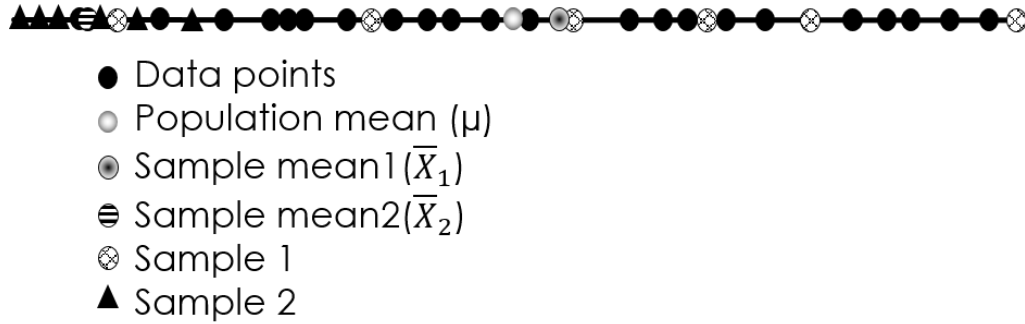


Figure 2.12: Variance for a population and a sample of a population.

As shown in Figure 2.13, the first sample (circles filled with crossed lines) will give a variance that is close to the population variance. Sample 2 (black triangles) variance will be around the sample mean 2 (stripped circle). To solve this issue we need to change the variance formula:

$$S^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{X})^2$$

By this, we will have the result of the summation for n samples of the difference between each point and its mean squared divided by n-1. This will make the variance around the population mean instead of being diverged away from the mean.

The below formulas show that variance can be written in different representations that will be used by us:

$$\begin{aligned} \sigma^2 &= \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2 \\ &= \frac{\sum_{i=1}^N (x_i^2 - 2x_i\mu + \mu^2)}{N} = \frac{\sum_{i=1}^N x_i^2}{N} - \frac{\sum_{i=1}^N 2x_i\mu}{N} + \frac{\sum_{i=1}^N \mu}{N} \end{aligned}$$

$$\begin{aligned}
&= \frac{\sum_{i=1}^N X_i^2}{N} - \frac{2\mu \sum_{i=1}^N X_i}{N} + \frac{\mu \sum_{i=1}^N 1}{N} \\
&= \frac{\sum_{i=1}^N X_i^2}{N} - 2\mu^2 + \mu \quad (\mu = E[X]) \\
&= E[X^2] - 2(E[X])^2 + E[X] \\
&= E[(X - E[X])^2]
\end{aligned}$$

### 2.6.1 Variance of Binomial Distribution

Variance of Binomial distribution is another way to find the distribution of certain values by finding the variance from its binomial distribution. Again, the result will tell us how the data are distributed around the mean. To find the variance of a binomial distribution, the result of the derivation below should be used.

The probability of binomial distribution =  $P(x = k) = \binom{n}{k} P^k (1 - P)^{n-k}$ ,

$$E(x) = n \cdot P$$

$$\sigma^2 = \sum_{k=1}^N (k - nP)^2 \cdot \binom{n}{k} \cdot P^k \cdot (1 - P)^{n-k} = -n(P^2 - P)$$

$$= -nP(P - 1)$$

$$= -nP(-1(1 - P))$$

$$= nP(1 - P)$$



## 2.7 Random (or Stochastic) Variable

Random variable is a possible value that represents the outcome of an experiment which has not yet happened. It is similar to the traditional variable ( $x+2=y$ , as we can solve for  $y$  by substituting  $x$ ) in a sense of referring to some values but actually not solving for them. A random variable is essentially a function that maps all points in the sample space to real numbers. For example, the continuous random variable  $X(t)$  might map time to position at any point in time,  $X(t)$  would tell us the expected position [6]. Or it is a function that maps from the world of random processes to actual numbers.

Let's take an example to clarify the meaning of a random variable:

I want to quantify if there is rain tomorrow.

$$X = \begin{cases} 150 & \text{Rain} \\ 20 & \text{No rain} \end{cases} \quad X: \text{ can be any number depending on how it is defined.}$$

this example shows the Random value as a process to maps us from a world of a process to a number and this number is random as we don't know if it is going to rain or not.

There are two types of random variables; discrete and continuous.

### 2.7.1 Discrete Random Variables

Discrete Random Variable is the same concept as in the example that we already discussed: flipping coin: when flip coin there are two outcomes either 1 or 0. When I through a dice, there are six discrete cases, which means there are no further results among these six results. In the raining example, there is either yes or no (Yes, it is going to rain or

No). We have here a countable number of outcomes. Figure 2.13 shows an example of discrete values of a six sides dice.

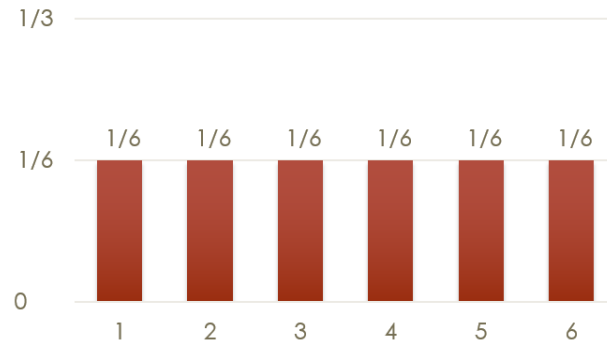


Figure 2.13: The probability distribution of all possible outcomes is discrete random values.

### 2.7.2 Continuous Random Variables

Continuous random variable can take infinite number of outcomes. An example can be the exact amount of rain! I can have 1", 1.0009" or 2.00112". Furthermore, we can have infinite number of numbers between any two numbers. The bell shape in Figure 2.14 shows a continuous distribution of the raining amount.

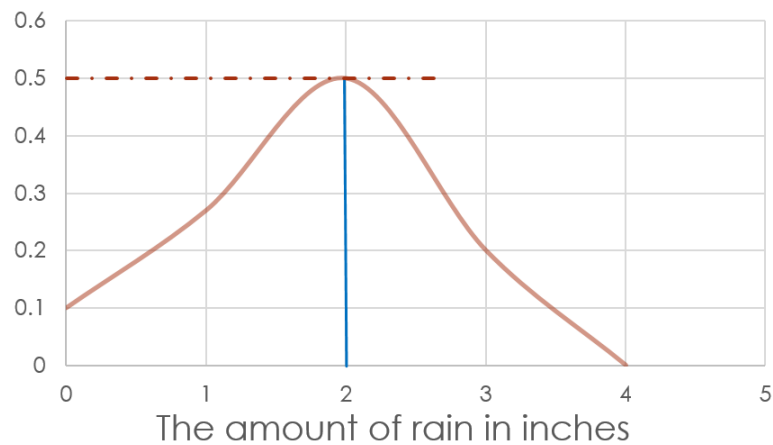


Figure 2.14: Continuous values representing the amount of raining.

What is the probability of having exactly 2” of rain? Logically, it is not possible to get exactly 2” (Not 2.0000001” or 1.9999999”!) as there is no way to measure an exact 2”! [7]. There is always more molecules in either side! Therefore,  $P(Y=2”) = 0$ . However, We can find the P(continuous random variable) by saying what is the probability of  $Y = \text{Almost } 2”$  which means  $P(|Y-2| < 0.1)$ . In other words, it is finding the area between 1.9 and 2.1 which is the definite Integral between 1.9 and 2.1 (Figure 2.15). The probability is the area under this curve. If the curve is defined as  $f(x)$ , then  $P(1.9 < Y < 2.1) = \int_{1.9}^{2.1} f(x) dx$ .

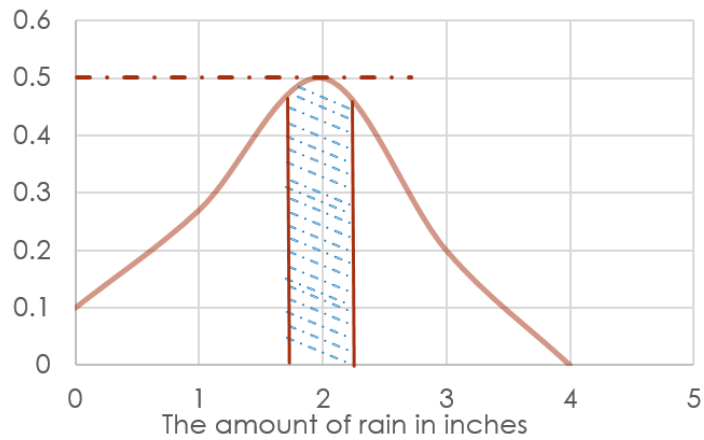


Figure 2.15: Finding the area between two sets (finding the probability of a range).

Therefore, the probability of an exact certain value is finding the area below the point (the shaded area in Figure 2.15) under the curve of the probability distribution which is 0.

Additionally, the probability of getting some result below a certain value is also applicable as shown in Figure 2.16 which is the probability that the rain will be below 1”.

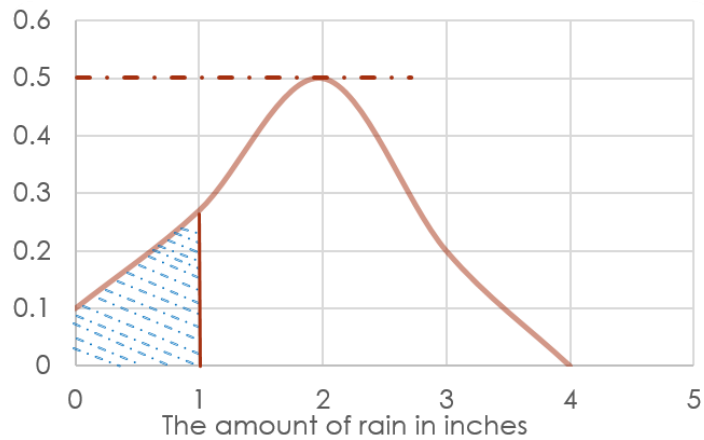


Figure 2.16: Finding the probability below a certain value.

## 2.8 Normal (or Gaussian) Distribution (Continuous)

The normal ( $z$ ) distribution is a continuous distribution that arises in many natural processes. "Continuous" means that between any two data values we could (at least in theory) find another data value. For example, men's heights vary continuously and are the result of so many tiny random influences that the overall distribution of men's heights in America is very close to normal. Another example is the exact amount of rain tomorrow. Figure 2.17 shows an example of normal distribution and cumulative distribution function [8].

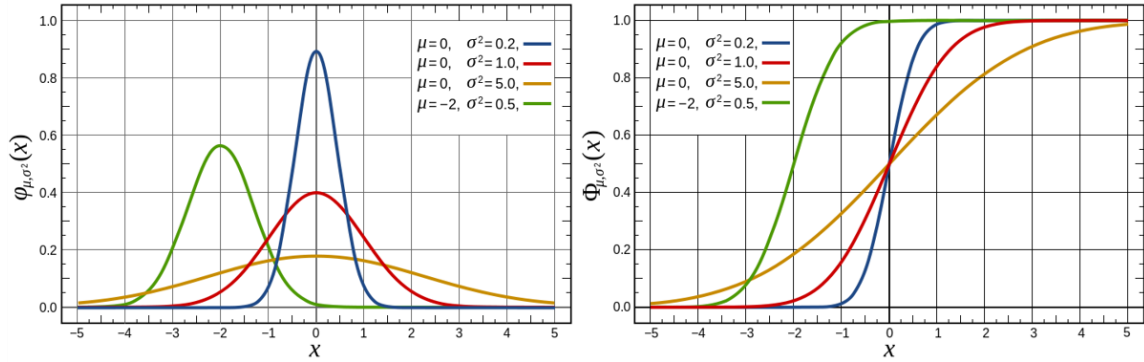


Figure 2.17: Normal distribution and Cumulative distribution function examples.

Is normal distribution symmetrical? Yes, fundamentally, the standard deviation and the mean values describe the bell shape of the normal distribution. The standard deviation grows on the right side and creates the positive values as well as the curve. On the mean's left side, the standard deviation reflects its positive values to represent the negative values and the bell curve on the negative side. Additionally, the mean, median<sup>1</sup>, and mode<sup>2</sup> of the distribution are equal and located at the peak (i.e., height of the curve).

The equation to find the normal distribution is listed below:

$$\text{Probability Distribution Function (PDF)} = f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

where  $\frac{x-\mu}{\sigma} = z$  which is called the standard score.

---

<sup>1</sup> Median is the number that comes in the middle after rearranging the analyzed numbers in order.

<sup>2</sup> Mode is the number that repeated more often than other numbers.

$$\text{PDF} = \frac{1}{\sqrt{2\pi\sigma^2}e^{z^2}}$$

$$\text{Cumulative Distribution Function (CDF)} = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{\sqrt{2\sigma^2}}\right)\right]$$

To find the probability for a certain outcome:

$$P(x) = \int_{x_1}^{x_2} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} dx$$

$$\text{CDF} = \int_{-\infty}^x \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} dx$$

### 2.8.1 Example

What is the percentage of men that are between 5'10" and 6'1" if men's heights in inches follow the  $N(69, 3)$  distribution?

Draw a sketch with the peak at 5'9" and the points of inflection of the bell-shaped curve at 5'6" and 6'0". We put the points of inflection 3 inches above and below the mean because we were given that the standard deviation was 3. Then shade the area between 5'10" and 6'1", and mark the answer (found by punching in normal cdf. Answer: 28%. Figure 2.18 and Figure 2.19 below show the results.

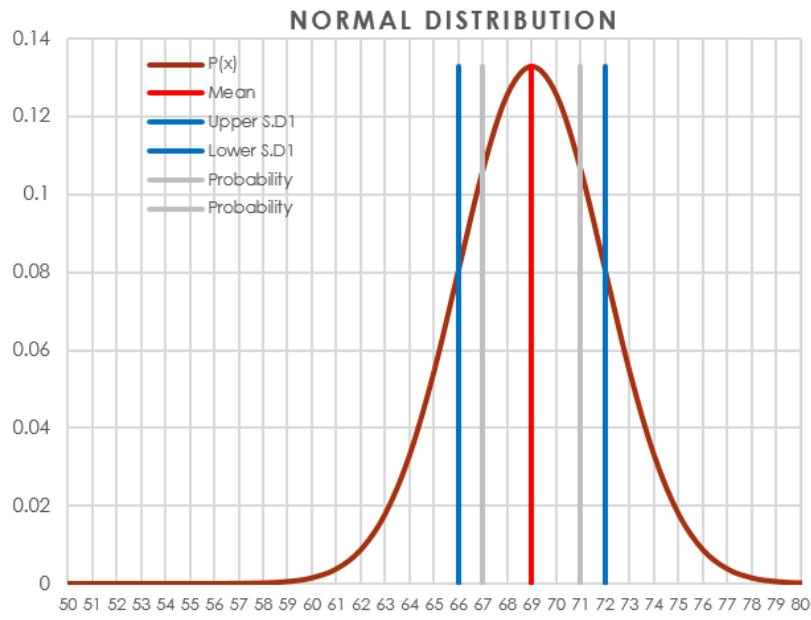


Figure 2.18: The normal distribution.

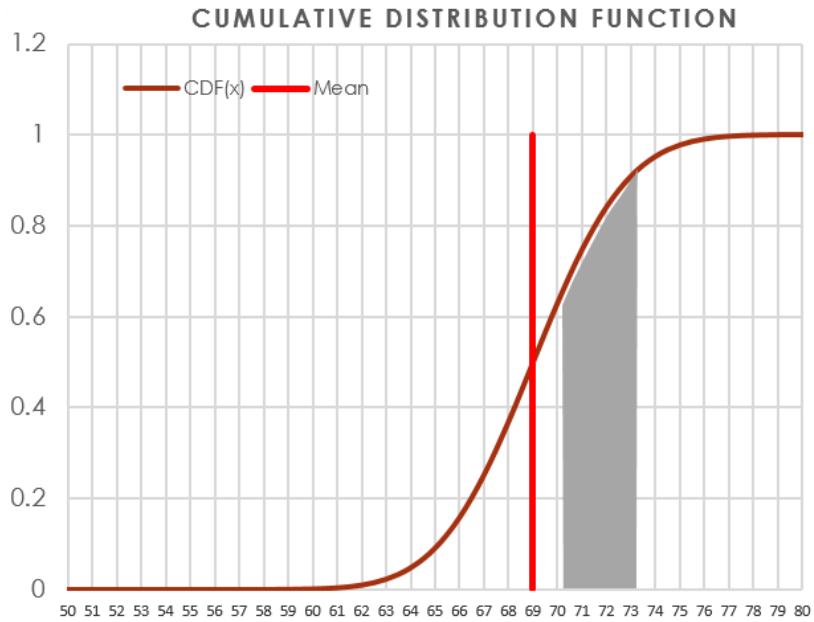


Figure 2.19: The cumulative distribution function.

### 2.8.2 Example

How tall must a man be at the 90th percentile of height? To find the solution, draw the curve as in Figure 2.19 (mean at 69, points of inflection at 66 and 72). Mark and shade 10% area in a right tail, or 90% area in a left tail. 72.84 inches. `NORMAL.INV` (Probability, Mean, Standard deviation) function is used in Excel to get the answer.

## 2.9 Normal Distribution and Binomial Distribution

As mentioned before, Binomial distribution represents the discrete outcomes like the probability of selecting faces of a six sides dice. On the other hand, the Normal distribution represents continuous outcomes like the normal distribution of the amount of rain for tomorrow. However, if we do the experiment for infinite number of times for many



different events, the normal distribution will diverge from the binomial distribution. A simple example is flipping a coin for thousand times and writing down how many times I got a head. If I repeat this process for an infinite number of times, I will get the normal distribution of getting heads after flipping a coin for thousand times. Figure 2.20 and Figure 2.21 below show the difference between Binomial distribution and normal distribution in different types of experiments. The first one is running the experiment for 4 times and the second one I run the same experiment for fifty times.

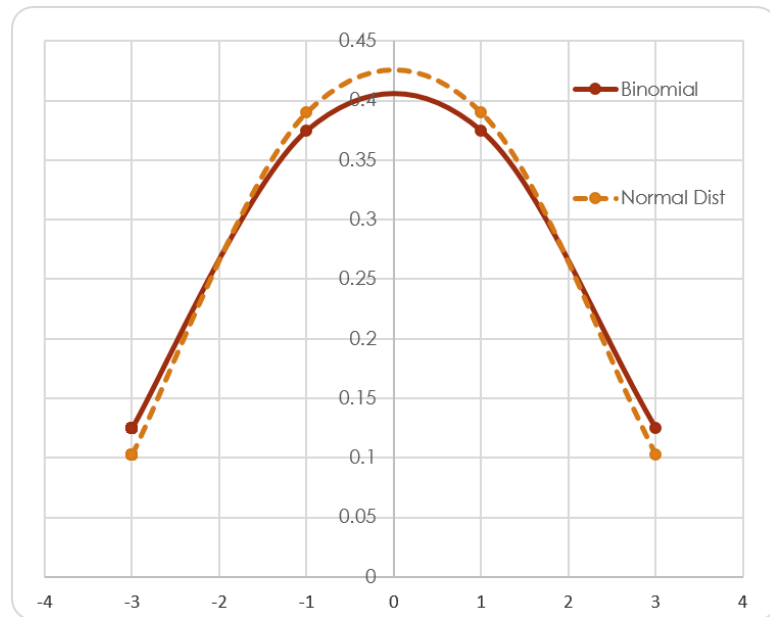


Figure 2.20: Binomial distribution.

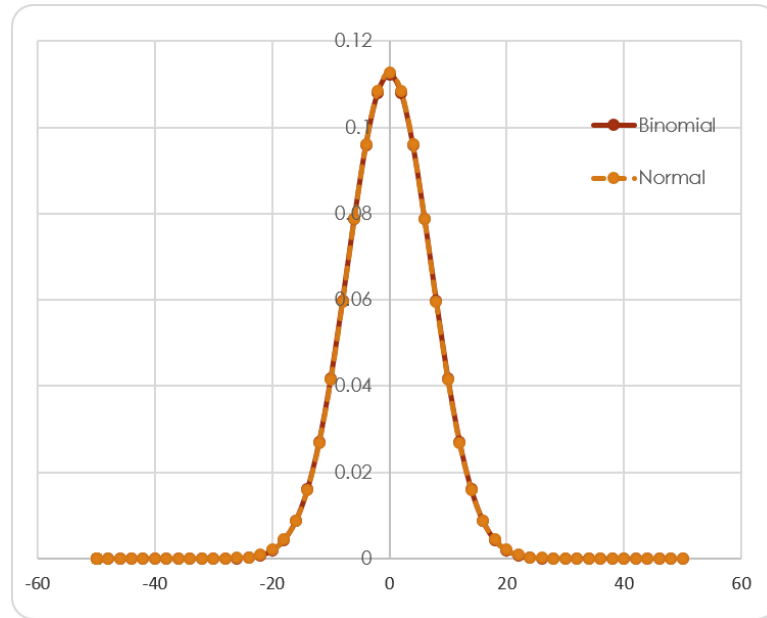


Figure 2.21: Normal distribution.

## 2.10 Compare Normal distribution to other Distribution Functions

Let's take some examples of samples and see whether they are normally distributed or not and give an explanation for the reason why not [9].

- Hand span of random undergrad students. There is a good normal distribution but the left end won't be in negative and the right end won't be a very high value (no one has a mile hand span or a negative hand span!).
- The annual salaries of a large company. We will have a right skewed distribution. There is a threshold from the left (the minimum wages) and there are two humps (means), one for most people and one for CEOs (Chief executive officer) Figure 2.22).

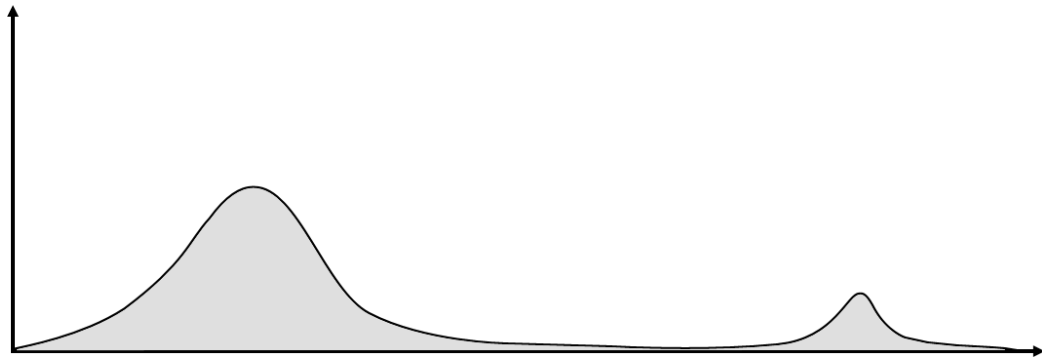


Figure 2.22: Double humped normal distribution.

- Figure 2.23 below shows the possibilities of having a sample of employees that have different annual salaries. There is no gap between the high salaries and the low ones. Therefore, the distribution will be positive skewed which means that there are low number of employees taking high salaries (based on their experience for instance) and the majority of them are taking low to medium annual salaries.



Figure 2.23: Positive skew distribution.

- The dates printed on 100 pennies taken from a cash drawer in a convenience store. We might have here a left skewed distribution as the majority will be a new pennies in 2009 for example and a long tails for older pennies (Figure 2.24).



Figure 2.24: Negative skewed distribution.

### 2.11 Normal Distribution Empirical Rule

68-95-99 is an empirical rule for the normal distribution. It means that 68% of the population is between the first standard deviation and 95% between the second standard deviation. Eventually, 99.7% of the population is between the third standard deviation. The idea of this rule is showed in Figure 2.25.

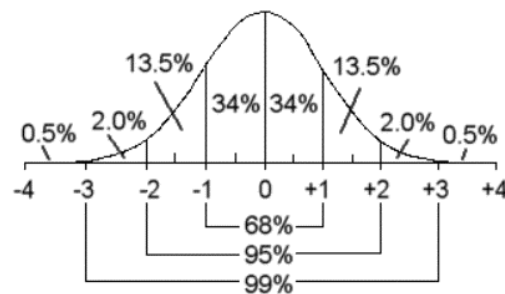


Figure 2.25: 68 95 99.7 rule [10].

### 2.11.1 Example

The drivers' speeds in Portland's highways are normally distributed with a mean of 55 m/h and standard deviation of 10 m/s. Estimate the cars speed percentage of the following (without using calculator):

- a) Less than 45 m/h
- b) Between 35 m/h and 75 m/h
- c) More than 85 m/h.

The solution simply achieved by using 68-95-99.7 rule.

- a)  $100 - (68/2) - 50 = 15.7\%$
- b) 95%
- c)  $100 - 50 - (99.7/2) = 0.15\%$

Figure 2.26 shows the normal distribution of the drivers' speed in Portland.

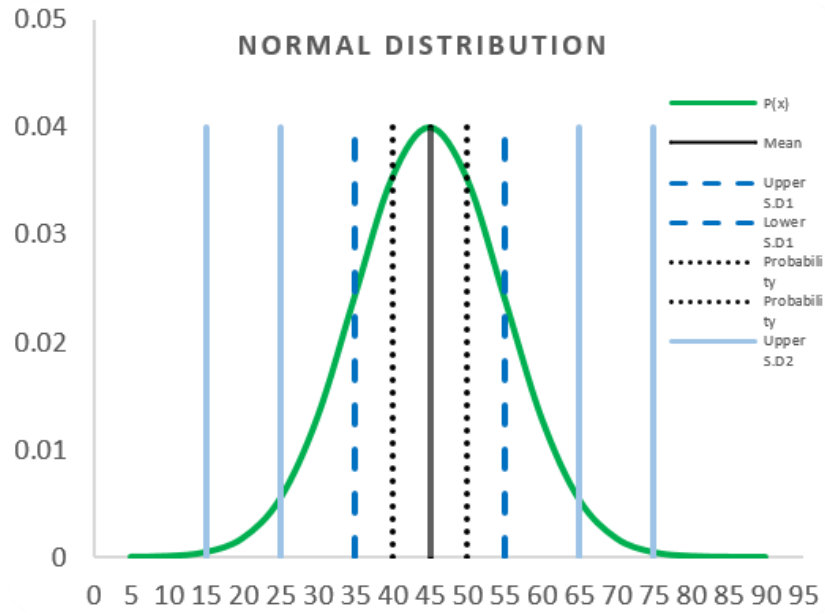


Figure 2.26: Normal distribution

Chapter 3: Introduction to Kalman Filter

### Chapter 3: Introduction to Kalman Filter

In this chapter, many basic filters will be discussed before going to the main topic which is Kalman Filter. There are many reasons to start with these filters. First, it is a good introduction to filters and why they are used. Second, it is more important to learn more about recursion used in various filters. Finally, it is a good introduction to Kalman Filter itself, because later on a comparison between these filters and Kalman Filter will show the powerful abilities of Kalman Filter. This chapter presents recursive filters in a successive explanation way provided with very straight-forward examples, which will allow us to move a step forward toward the explanation of the Kalman Filter.

What is recursion? It is a method of reusing the previous result in the current process. It has a good efficiency in computation (as we will see in this section). Also, it has an advantage in the aspect of memory storage as the recursive approach needs only the previous result and not all the previous quantities. Moreover, recursion is applicable in simple filters like average filter, and in more complicated filters like the Kalman Filter.

#### 3.1 Average Filter

Average filter is a very simple filter used to average the acquired data in order to eliminate the noises. The main average equation is:

$$\bar{X}_k = \frac{X_1 + X_2 + \dots + X_k}{k} \quad (\text{Equation 3.1})$$

This is the traditional function for finding the average of certain values, where  $k$  is the size of acquired data and  $\bar{X}_k$  is the average value. However, applying this equation will be



expensive as we need a bigger memory to save the values when the number of values grows every time a new datum is introduced. Additionally, a bigger processor is needed as the calculations will get bigger when more data is acquired. Using recursion here will be effective as we need only the previous average and the new datum. Currently, we need to derive the recursive average filter.

Recursive average filter derivation:

$$\bar{X}_{k-1} = \frac{X_1+X_2+\dots+X_{k-1}}{k-1} \text{ (Equation 3.2) , multiplying Equation 3.1 by k we get}$$

$$k\bar{X}_k = X_1 + X_2 + \dots + X_k$$

Dividing both sides by k-1:  $\frac{k\bar{X}_k}{k-1} = \frac{X_1+X_2+\dots+X_k}{k-1}$ , the right side could be written as

$$\frac{k\bar{X}_k}{k-1} = \frac{X_1+X_2+\dots+X_{k-1}}{k-1} + \frac{X_k}{k-1}, \text{ The bold part can be substituted by } \bar{X}_{k-1} \text{ in Equation 3.2.}$$

$$\frac{k\bar{X}_k}{k-1} = \bar{X}_{k-1} + \frac{X_k}{k-1}$$

$$\bar{X}_k = \frac{k-1}{k} \bar{X}_{k-1} + \frac{1}{k} X_k$$

$$\bar{X}_k = \alpha \bar{X}_{k-1} + (1-\alpha)X_k, \text{ where } \alpha = \frac{k-1}{k}, \frac{1}{k} = 1-\alpha$$

This is the main function of a recursive average filter.

### 3.1.1 Example

Average Filter is tested here with a real data that comes from one of the sonars that is used in MCEC-Bot. This data has some false measurements which are some zeros

showing up every while. We got this kind of measurements after the improvements of having the chaining method. More information about this measurements and the chaining method can be found in the ultrasonic section in the fifth chapter. Anyway, this example will show that Average filter has very limited applications. There is accumulated error and diverge due to the effect of the earliest measurements. However, it might be a good sensor for a fixed state. In other word, this filter may be good if the robot is stationary, but it is not the case all the time.

Figure 3.1 shows the code of the average filter algorithm. It receives the sonar measurements and returns the average of the measurements.

AvgFilter
<pre>function avg = AvgFilter(x) persistent prevAvg k persistent firstRun if isempty(firstRun)     k = 1;     prevAvg = 0;     firstRun = 1; end alpha = (k - 1) / k; avg = alpha*prevAvg + (1 - alpha)*x; prevAvg = avg; k = k + 1;</pre>

Figure 3.1: Average filter equation.

Figure 3.2 shows GetMaxSonar1 code that reads the sonar from a file that has a real data taken from the front sonar of the MCECS-Bot.

```

GetMaxSonar1
function h = GetMaxSonar1()
%
%
persistent son1    % SonartAlt.mat
persistent k firstRun

if isempty(firstRun)
    load Son1
    k = 1;

    firstRun = 1;
end

h = son1(k);

k = k + 1;

```

Figure 3.2: Get Sonar.

Figure 3.3 shows the main code (ResultAveSonar1) that gets the sonar measurements and passes them to AvgFilter(). It presents the average on a graph, which is Figure 3.4.

```

ResultAveSonar1
clear all

Nsamples = 500;
Avgsaved = zeros(Nsamples, 1);
Xmsaved = zeros(Nsamples, 1);

for k=1:Nsamples
    xm = GetMaxSonar1();
    avg = AvgFilter(xm);

    Avgsaved(k) = avg;
    Xmsaved(k) = xm;

```

```
end

dt = 1;
t = 0:dt:Nsamples-dt;

figure
plot(t, Xmsaved, 'r')
hold on
plot(t, Avgsaved, ':', 'LineWidth', 2)
xlabel('Time (sec)');
ylabel('Distance (inch)');
legend('Measured', 'Estimated')
```

Figure 3.3: Test Average filter.

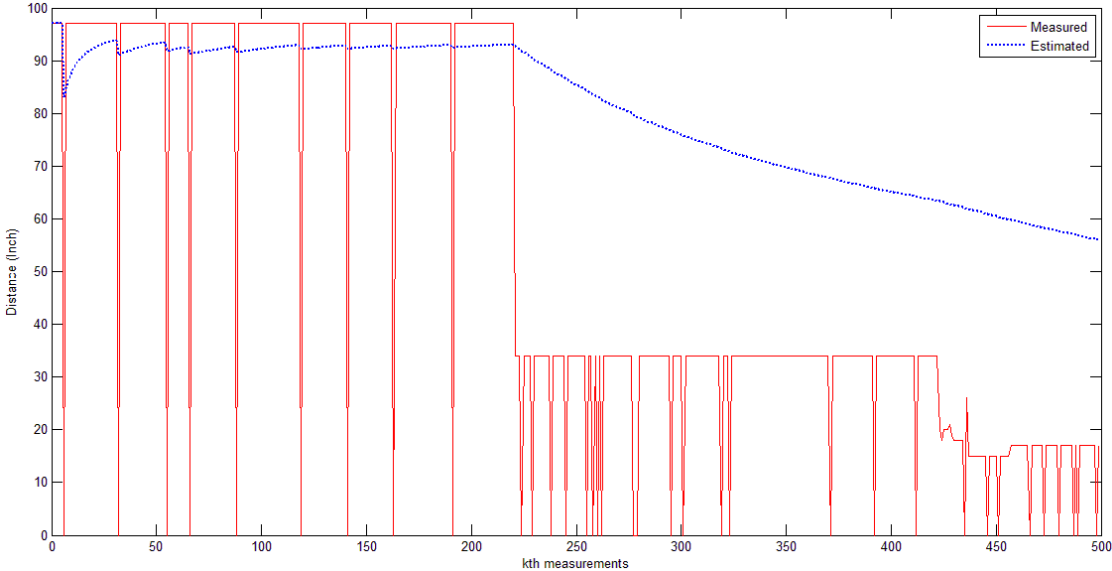


Figure 3.4: Average filter results.

It is obvious in Figure 3.4 that the Average Filter lost tracking the signal after it changed at 225. This shows how that filter’s results get diverged by the effect of the earliest measurements.

## 3.2 Moving Average Filter (MAF)

A “Moving Average Filter” is the second filter presented in this chapter. This filter is used to eliminate the noise by calculating the average of the last  $n$  measurements. Average filter doesn't show the changing pattern of the measurements as it averages all acquired data. The Moving Average Filter makes only the last  $n$  measurements to contribute in the new estimation. This will keep the pattern of the changing measurements. The simple average filter is not efficient and not very descriptive to our measured data because all the dynamic changes are being removed and only a signal value represents the whole data. A Moving Average Filter is used in this case as the observed data are changing along with time. Only a certain number of measurements will be chosen. Each time we measure a datum the set of data will be “*shifted*”.

$$\bar{X}_k = \frac{X_{k-n+1} + X_{k-n+2} + \dots + X_k}{n}, \bar{X}_k \text{ is the average from } k-n+1^{th} \text{ to } k^{th}.$$

$$\bar{X}_{k-1} = \frac{X_{k-n} + X_{k-n+1} + \dots + X_{k-1}}{n}, \text{ The moving average of the previous datum.}$$

Subtracting these two equations we get the Moving Average Filter in its recursive form:

$$\bar{X}_k = \bar{X}_{k-1} + \frac{X_k - X_{k-n}}{n} \quad (\text{Equation 3.3})$$

### 3.2.1 Example

Moving Average Filter is used in this example to filter out a real data taken from front sonars of the MCECS-Bot. The sonar used to measure a distance from a wall gives a measurement every 50  $\mu$  second. The measurement was very noisy due to the acoustic

noise, which is the interference that was coming from other sonars. We faced this problem of noisy measurements before the improvements of having the chaining method (for more information refer to the ultrasonic sensors section in the fifth chapter). However, the noisy data is good to test the tolerance of each filter.

Figure 3.5 shows the algorithm of moving average filter. This function receives the sonar data from the main code (Figure 3.7) and returns the average value. The function that reads the real sensor data is shown in Figure 3.6.

```
Moving Average Filter
function avg = MovAvgFilter(x)
%
persistent prevAvg n xbuf
persistent firstRun

if isempty(firstRun)
    n = 20;
    xbuf = x*ones(n+1, 1);

    k = 1;
    prevAvg = x;

    firstRun = 1;
end
for m=1:n
    xbuf(m) = xbuf(m+1);
end
xbuf(n+1) = x;

avg = prevAvg + (x - xbuf(1)) / n;
prevAvg = avg;
```

Figure 3.5: Moving Average Filter code.

The result of the second filter ( $n = 20$ ) in Figure 3.8 is coming from MovAvgFilter3 function. This function is similar to MovAvgFilter with only one difference which is  $n = 20$  instead of 5.

```
Sonar1
function h = Sonar1()
%
persistent son    % Son.mat
persistent k firstRun

if isempty(firstRun)
    load son
    k = 1;
    firstRun = 1;
end
h = son(k);
k = k + 1;
```

Figure 3.6: Get Sonar code.

```
ResultMovAvSonar
clear all

Nsamples = 500;
Xsaved = zeros(Nsamples, 1);
Xsaved2 = zeros(Nsamples, 1);
Xmsaved = zeros(Nsamples, 1);

for k=1:Nsamples
    xm = Sonar1();
    x = MovAvgFilter(xm);
    x2 = MovAvgFilter3(xm);
    Xsaved(k) = x;
    Xsaved2(k) = x2;
    Xmsaved(k) = xm;
end

dt = 0.02;
```

```

t = 0:dt:Nsamples*dt-dt;

figure
plot(t, Xmsaved, 'r. ');
hold on
plot(t, Xsaved2, 'black');
plot(t, Xsaved, 'b');
xlabel('Time (sec)');
ylabel('Distance (inch)');
legend('Measured', 'Estimated')
%legend('Measured', 'Moving average')
legend('Measured','n = 5', 'n = 20')% when the no. of datum is
bigger the filter will
        %eliminate more noise but it is slower
        %Smaller n means sharper/faster filter but more
noise.

```

Figure 3.7: Test Average filter code.

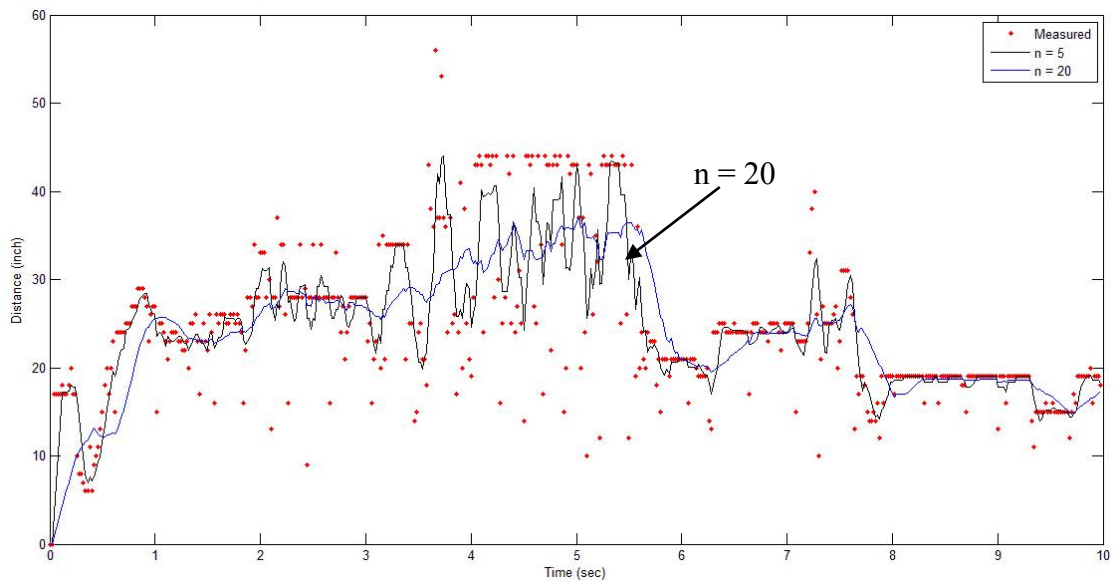


Figure 3.8: Moving Average filter result.

Figure 3.8 above shows the difference between the results when the size of the last measurements is different. When  $n = 5$  the estimation is sharper and more descriptive with



the changing values. On the other hand, when  $n = 20$ , the estimation is smoother and less affected by the noise. There are both advantages and disadvantages of choosing either bigger or smaller  $n$ . Another shortcoming of the Moving Average Filter is that it is not able to reduce noise and it tracks the changes in the measurements. The filter in general is unable to distinguish which particular measurement contributes more than the others. In other words, the filter doesn't answer this question; "is the current measurement more important than the previous one or not?" This answer becomes more obvious after expanding equation 3.4 to the following form:

$$\bar{X}_k = \frac{1}{n}\bar{X}_{k-n+1} + \frac{1}{n}\bar{X}_{k-n+2} + \dots + \frac{1}{n}\bar{X}_k \quad (\text{Equation 3.4})$$

Each measured datum has the same weight ( $1/n$ ). A good example to understand the difference is that there is a robot moving toward a wall and it takes measurements. The very recent measurement is more important than the old ones, hence the old measurements will weakly affect the robot perception.

### 3.3 Low Pass Filter (LPF)

Simply, Low pass filter is a filter that passes signals with low frequencies and blocks higher frequencies, as usually the noise is in a higher frequency than the main signal. In LPF  $\bar{X}_k$  is estimate instead of average. The LPF equation is:

$$\bar{X}_k = \alpha \bar{X}_{k-1} + (1-\alpha)z_k, \quad 0 < \alpha < 1.$$

Let's analyze how the LPF is different from the Moving Average Filter:

$\bar{X}_{k-1} = \alpha \bar{X}_{k-2} + (1-\alpha)X_{k-1}$ , substitute in the original equation of moving average filter:

$$\bar{X}_k = \alpha \{ \alpha \bar{X}_{k-2} + (1-\alpha)X_{k-1} \} + (1-\alpha)X_k$$

$$= \alpha^2 \bar{X}_{k-2} + \alpha (1-\alpha)X_{k-1} + (1-\alpha)X_k \quad (\text{Equation 3.5})$$

If the same thing is done with  $\bar{X}_{k-2} = \alpha \bar{X}_{k-3} + (1-\alpha)X_{k-2}$  and substitute it in Equation 3.5, the final result is:

$$\bar{X}_k = \alpha^3 \bar{X}_{k-3} + \alpha^2 (1-\alpha)\bar{X}_{k-2} + \alpha (1-\alpha)\bar{X}_{k-1} + (1-\alpha)z_k$$

The last equation shows how the weight is different between each measurement based on precedence. The last equation shows how the LPF deals differently with the acquired measurements. In LPF, the smaller  $\alpha$  is the more dependent  $\bar{X}_k$  on the current measurement gets, which is illustrated below:

$$\bar{X}_k = \alpha \bar{X}_{k-1} + (1-\alpha)z_k$$

### 3.3.1 Example

The same example used for the Moving Average Filter will be used here. Figures below (Figure 3.9, Figure 3.10, Figure 3.11 and Figure 3.12) show Matlab code and the filter results.

<pre> LPF function xlpf = LPF(x) % % persistent prevX persistent firstRun  if isempty(firstRun)     prevX = x; </pre>
---

```

    firstRun = 1;
end

alpha = 0.7;
xlpf = alpha*prevX + (1 - alpha)*x;

prevX = xlpf;

```

Figure 3.9: LPF code.

In the LPF results, LPF and LPF2 are presented. LPF2 is basically identical to LPF (Figure 3.9) but with different alpha value in order to show the difference between the two results. Each time there is a measurement, it will be passed from the main code to the LPF function in order to get the estimated result. The measurement value is obtained by calling the Sonar1() function by the main code ResultLPFSonar which is shown in Figure 3.11.

```

Soanr1
function h = Sonar1()
persistent son % son.mat
persistent k firstRun

if isempty(firstRun)
    load son
    k = 1;
    firstRun = 1;
end
h = son(k);
k = k + 1;

```

Figure 3.10: Get Sonar code.

Get Sonar hasn't changed. Simply it just loads a file (son.mat) which includes the measurements. The main function (Test LPF) will call this function in every measurement step in order to simulate the sonar measurements.

```

ResultLPFSonar
clear all

Nsamples = 500;
Xsaved = zeros(Nsamples, 1);
Xsaved2 = zeros(Nsamples, 1);
Xmsaved = zeros(Nsamples, 1);

for k=1:Nsamples
    xm = Sonar1();
    x = LPF(xm);
    x2 = LPF2(xm);
    Xsaved(k) = x;
    Xsaved2(k) = x2;
    Xmsaved(k) = xm;
end

dt = 0.02;
t = 0:dt:Nsamples*dt-dt;

figure
plot(t, Xmsaved, 'r. ');
hold on
plot(t, Xsaved2, 'b');
plot(t, Xsaved, 'k');
%legend('Measured', 'LPF')
legend('Measurements', 'ALpha = 0.9', 'ALpha = 0.3')
xlabel('Time (sec)')
ylabel('Distance (inch)')

```

Figure 3.11: The main code that gets the real data from Sonar1() and sends it to LPF(). Eventually, it shows the result in Figure 3.12.

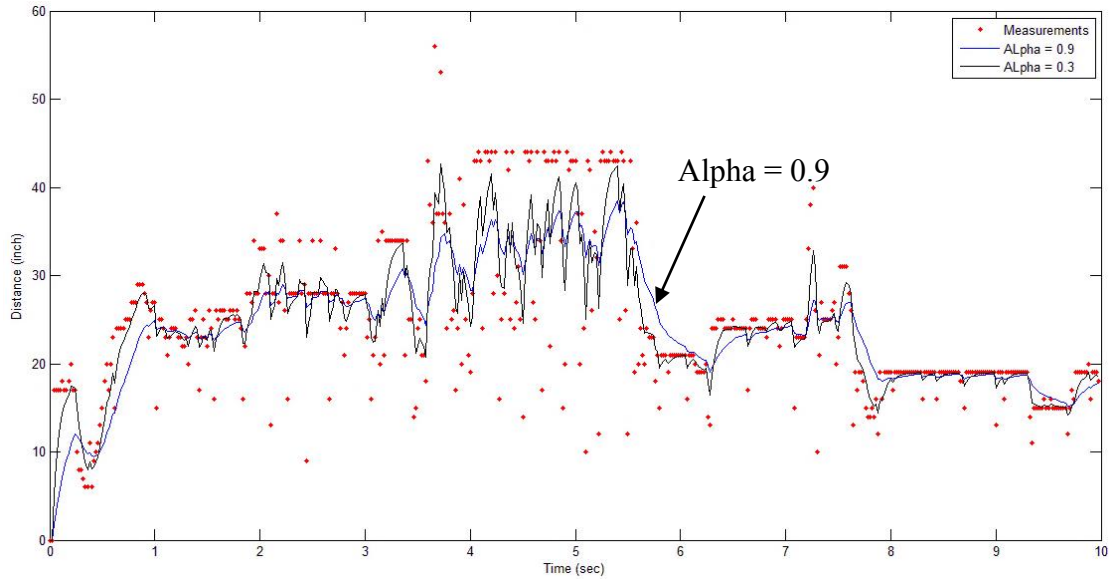


Figure 3.12: LPF results.

Figure 3.12 shows the results of two different alphas. Going back to the LPF equation  $\bar{X}_k = \alpha \bar{X}_{k-1} + (1 - \alpha)X_k$ , it is obvious that there is a different contribution of the measurements to the result when  $\alpha$  equals 0.3 or 0.9. When  $\alpha$  was smaller, it made the new measurement contributed more to the new estimation. On the other hand, when  $\alpha$  equaled 0.9, the old estimate had more weight and the estimation didn't change by the new measurements. As shown in the LPF results, a smaller alpha means sharper and faster results. Contrary, a bigger alpha caused a delay in the output estimate but the estimation was smoother.

Chapter 4: Kalman Filter

## Chapter 4: Kalman Filter

After the mathematical and some filter theory introduction, now it is time to present the Kalman Filter. A Kalman Filter is an optimal estimator – i.e. it infers parameters of interest from indirect, inaccurate and uncertain observations. It is recursive so that new measurements can be processed as they arrive [11]. In what sense the Kalman Filter is optimal? If all noise is Gaussian, the Kalman Filter minimizes the mean square error of the estimated parameters. What if the noise is NOT Gaussian? Assuming that the noise is Gaussian, one needs only mean and standard deviation of noise and the Kalman Filter is the best linear estimator. Non-linear estimators may be better for non-Gaussian noise. This leads for instance to Extended Kalman Filter. The steps of KF are shown in Figure 4.1.

Why is Kalman Filtering so popular?

- Good results in practice due to optimality and structure.
- Convenient form for online real time processing.
- Easy to formulate and implement given a basic understanding [11].
- Based completely on algebra.

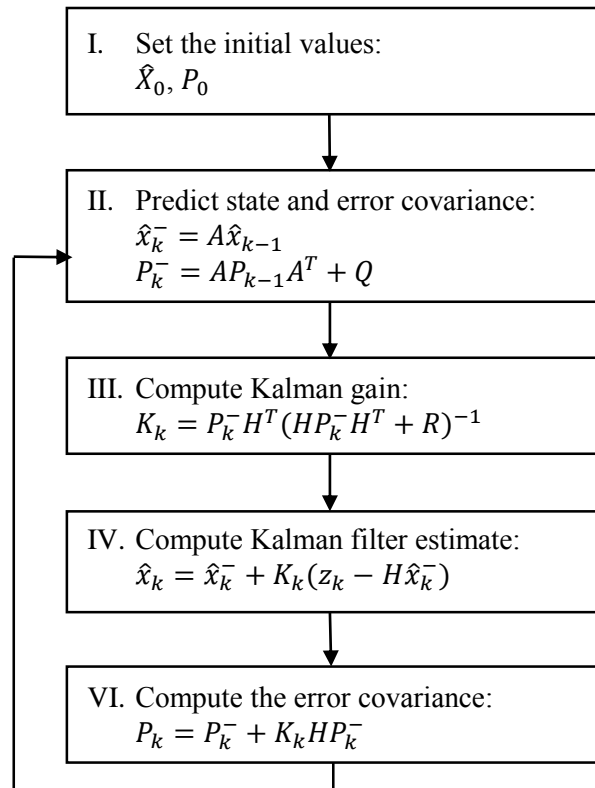


Figure 4.1: Kalman Filter Algorithm [12].

Kalman Filter algorithm is shown in the Figure 4.1 above. Briefly, Kalman Filter steps are explained as follows:

- I. Set the initial values of  $\hat{X}_0$  (the value that we made Kalman Filter for) and  $P_0$  which is the error covariance. The concept of the error covariance is explained in section 4.2.
- II. Based on the initialized values (first time) or based on the previous estimated values, the predicted values will be calculated.
- III. Based on the predicted values from the second step, the Kalman gain  $K$  is calculated. For now, Kalman gain  $K$  is similar to alpha in LPF.
- IV. Now, there is a new measurement and Kalman gain is computed. Current step is the step to compute the estimated value.
- V. The error covariance is computed.

The subscript  $k$  is just telling us that Kalman is a recursive process. Likewise, the subscript “-” means predicted value. Table 4.1 shows the Kalman Filter variables:

External input	$z_k$ (measurement)
Final output	$\hat{X}_k$ (estimate)
System model	A, H, Q, R
For internal computation	$\hat{X}_k^-, P_k^-, P_k, K_k$

Table 4.1: Kalman Filter variables.



System model variables (A, H, Q and R) are variables that are not computed inside Kalman Filter. They are set by the user based on the system characteristics and the purpose of Kalman Filter. The performance of Kalman Filter will be determined by these factors.

Kalman Filter could be summarized as following:

1. Predict state and error covariance for next time point, based on system model ( $A$  and  $Q$ ):  $\hat{x}_k^-$  and  $P_k^-$ , respectively.  $\hat{x}_k^-$  is the predicted state and  $P_k^-$  is the predicted error covariance.
2. Compensate the difference between measurement and prediction, and compute new estimate. This estimate is the final result of Kalman Filter:  $\hat{x}_k$  and  $P_k$ . Here  $\hat{x}_k$  is called the estimated state and  $P_k$  is the (new) estimate error covariance.
3. Loop through those two steps above [12].

#### **4.1 A Comparison between Kalman Filter and Low Pass Filter**

Comparing Kalman Filter to other filters is important as it helps us to have a better understanding of its features as well as it will help us to appreciate Kalman Filter breadth of applications and its power. In this comparison, it is obvious that Kalman Filter is quite similar to LPF. The following points show the similarities and differences:

- There is a relation between estimation in Kalman Filter and LPF<sup>3</sup>.

For Kalman Filter we calculate:

$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$ ,  $z_k$  is measurement and  $\hat{x}_k^-$  is prediction.

$$\hat{x}_k = \hat{x}_k^- (I - K_k H) + K_k z_k \quad \text{Equation 4.1}$$

If we assume that H is an identity matrix  $I$  the above equation becomes:

$$\hat{x}_k = \hat{x}_k^- (I - K_k) + K_k z_k$$

For LPF we calculate:

Substituting  $\alpha$  with  $1-K$  in the LPF equation ( $\bar{X}_k = \alpha \bar{X}_{k-1} + (1-\alpha)X_k$ ) we will get:

$$\bar{X}_k = (1 - K)\bar{X}_{k-1} + KX_k \quad \text{Equation 4.2}$$

Now compare equation 4.1 with equation 4.2. Both LPF and Kalman Filter are quite similar with some differences:

- Kalman Filter uses the previous prediction not estimate.
- Kalman Filter's gain  $K_k$  is changing every time; not fixed like alpha ( $\alpha$ ) in LPF.

---

<sup>3</sup> The way I derived KF and LPF equations in order to compare them is taken from "Kalman Filter for Beginner" [12].

- $K_k$  is a gain used as weighting in the above equation. The gain being computed is the significant difference between LPF and KF. Kalman gain equation:

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

- Difference between estimation and prediction:

In LPF ( $\bar{X}_k = \alpha \bar{X}_{k-1} + (1 - \alpha) X_k$ ) the previous estimate  $\bar{X}_{k-1}$  is used without being changed as shown in the Figure 4.2 below:

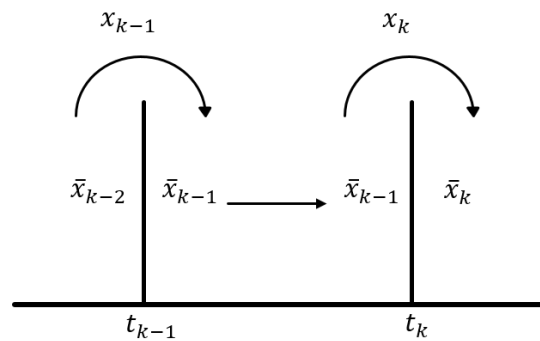


Figure 4.2: Estimation process in LPF [12].

In Kalman Filter the previous estimation is not used in the next state estimate calculation in contrast to LPF. More about Kalman Filter prediction process will be found in section 4.3.

## 4.2 Error Covariance

The equation to compute the error covariance is:

$$P_k = P_k^- - K_k H P_k^-$$

Error covariance is a matrix that indicates the difference between the estimate from Kalman Filter and the true unknown value. There is a relationship between  $x_k$  and its estimate  $\hat{x}_k$  and the error covariance  $P_k$ . This relation is explained in the formula 4.1:

$$x_k \sim N(\hat{x}_k, P_k) \quad (\text{Equation 4.3})$$

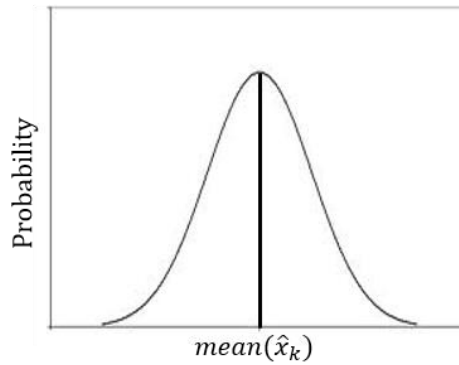


Figure 4.3:  $x_k \sim N(\hat{x}_k, P_k)$ .

The formula (Equation 4.1) tells us that there is a normal distribution of all possible values taken by  $x_k$  so that they are distributed as a bell shape (Figure 4.3). The width of this bell shape is determined by  $P_k$ . When the bell is wide the estimate will be less accurate than for a narrower bell. Simply, use  $\hat{x}_k$  and  $P_k$  to draw the bell and choose  $x_k$  with the highest probability. Error covariance is the mean of the square of the error estimate (The difference between the estimate  $\hat{x}_k$  and the true but unknown value  $x_k$ ). Additionally, the size of error is proportional to the size of the error covariance.

### 4.3 Prediction Process

As mentioned before, estimation from the previous step is used to compute the prediction of the next step's outcome.  $\hat{x}_k$  is obtained and a prediction needed to the next estimate value is  $\hat{x}_{k+1}$ .

Prediction equations are (one for estimate and the other one for error covariance):

- $\hat{x}_{k+1}^- = A\hat{x}_k$
- $P_{k+1}^- = AP_kA^T + Q$

$\hat{x}_k$  and  $P_k$  (in the above two equations) are calculated in the last two steps of Kalman Filter algorithm.  $A$  and  $Q$  are defined by the system model. ( $A$  and  $Q$  are used in prediction, and  $H$  and  $R$  are used in the estimation).

In Kalman Filter the estimate is  $\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$ , where  $\hat{x}_{k-1}$  is not shown here.  $\hat{x}_k^-$  is presented instead (The prediction of the current time point). The previous estimate is transferred into the current point prediction using the following equation:

$$\hat{x}_k^- = A\hat{x}_{k-1}$$

Let's substitute the above equation in the estimate equation:

$$\hat{x}_k = A\hat{x}_{k-1} + K_k(z_k - HA\hat{x}_{k-1})$$

Kalman Filter is different than LPF. There is one more step between the previous estimate and the current estimate in Kalman Filter while the previous estimate is used directly to find the new estimate in LPF. As a result we will have two different values for the same

point at the same time (in KF). Figure 4.4 illustrates the prediction process in Kalman Filter.

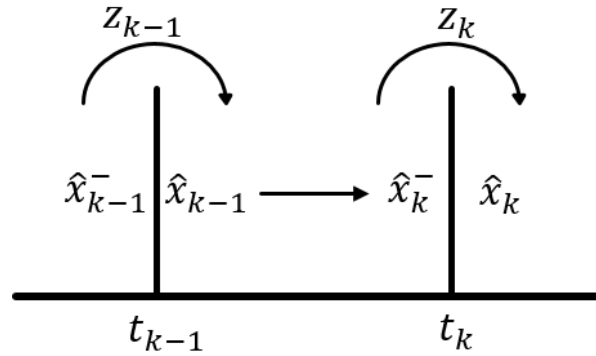


Figure 4.4: Prediction process in Kalman Filter. Simply, the previous estimation  $\hat{x}_{k-1}$  is not used in the process of estimating the current state. Instead, the previous estimation is converted to a prediction  $\hat{x}_k^-$  using the prediction equation:  $\hat{x}_k^- = A\hat{x}_{k-1}$ . Then the prediction of the current state is used to calculate the current estimation [12].

The state space model represents a physical system as a set of  $n$  first-order differential equations. This form is better suited for computer simulation than a single  $n$ th order input-output differential equation. From the system model, A and H matrices are acquired.

Part of state space representation:

1. **State Variables:** a subset of system variables which is known at an initial time  $t_0$  along with subsequent inputs are determined for all time  $t > t_0$ .
2. **State Equations:**  $n$  linearly independent first order differential equations relating the first derivatives of the state variables to functions of the state variables and the inputs.
3. **Output equations:** algebraic equations relating the state variables to the system outputs.

Below are basic state space examples. However, when we create system equations for our systems, we pick one of the following equations for capacitor, inductor and resistor, as convenient:

- Capacitor: the main equations are current and voltage equation.

$$V_C = \frac{1}{c} \int i_c dt, \quad i_c = c \frac{dV_C}{dt}$$

- Inductor: the inductor equations are:

$$V_L = L \frac{di_L}{dt}, \quad i_L = \frac{1}{L} \int V_L dt$$

- Resistor: it has only simple equation:

$$V_R = i_R \cdot R, \quad i_R = \frac{V_R}{R}$$

In this chapter, an introduction to the state space model has been presented side by side with examples. This introduction will help later to understand the process of deriving the system model for Kalman Filter.

#### 4.4 State Space Representation

Generally, there are two equations; one to describe the state equation and the other one for the output equation.

$$\dot{x} = Ax + Bu, \text{ The state equation.}$$

$$y = Cx + Du, \text{ The output equation.}$$

This is the standard form of representing all linear physical systems.

- $x$  is the state of the system. It is  $n \times 1$  matrix.
- $u$  is the input to the system. It is  $p \times 1$  matrix.
- $\dot{x}$  is the derivative of  $x$ . It is  $n \times 1$  matrix. It is like a storage for the previous states.
- $A$  is the state matrix of the system and it holds the property of the system ( $n \times n$  matrix).
- $B$  is the input matrix ( $n \times p$  matrix).
- $y$  is the output variable ( $q \times 1$  matrix).
- $C$  maps system state to the output ( $q \times n$  matrix).
- $D$  maps the input to the output ( $q \times p$  matrix).

The matrices form of the state space model equations is shown below. Figure 4.5 shows the block diagram of the state space model as it represents the state equation and the output equation.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{np} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} d_{11} & \cdots & d_{1p} \\ \vdots & \ddots & \vdots \\ d_{n1} & \cdots & d_{np} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \end{bmatrix}$$



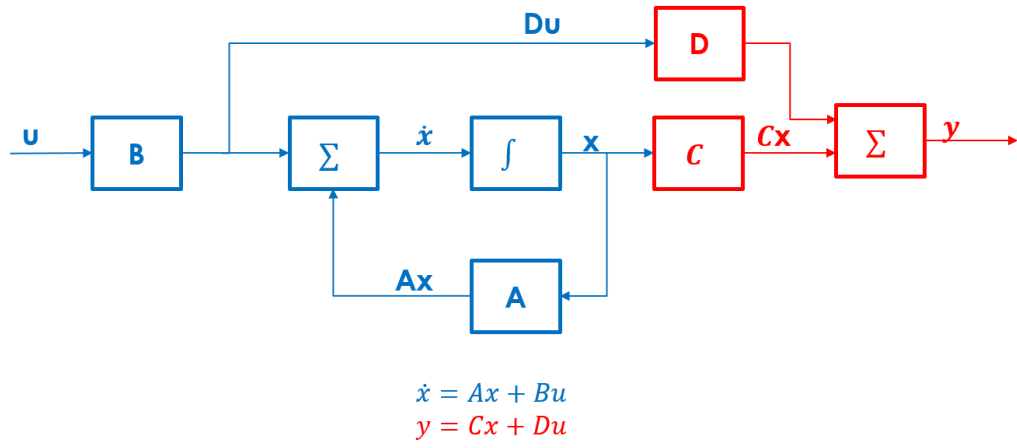


Figure 4.5: State space equations diagram.

## 4.5 State Space Property

State space model has properties as follows:

- Additivity: if a state ( $x_3$ ) equals the summation of two states ( $x_1 + x_2$ ) and the input to this state ( $u_3$ ) equals both inputs of these two states ( $u_1 + u_2$ ), the output of state  $x_3$  is:  $y_3 = y_1 + y_2$ .

$$\left. \begin{array}{l} x_1(t_0) \\ u_1(t) \quad t \geq t_0 \end{array} \right\} y_1(t) \quad t \geq t_0$$

$$\left. \begin{array}{l} x_2(t_0) \\ u_2(t) \quad t \geq t_0 \end{array} \right\} y_2(t) \quad t \geq t_0$$

$$\left. \begin{array}{l} x_3(t_0) = x_1 + x_2 \\ u_3(t) = u_1(t) + u_2(t) \quad t \geq t_0 \end{array} \right\} y_3(t) = y_1(t) + y_2(t) \quad t \geq t_0$$

- Homogeneity: if the initial state and the input signal are multiplied with a constant  $\alpha$ , the result should be multiplied by the same constant  $\alpha$ .

$$\left. \begin{array}{l} \alpha x_1(t_0) \\ \alpha u_1(t) \quad t \geq t_0 \end{array} \right\} \alpha y_1(t) \quad t \geq t_0$$

- Zero state response: it tells us what the system response is when the initial state equals zero and there is a certain signal applied.

$$\left. \begin{array}{l} x_1(t_0) = 0 \\ u_1(t) \quad t \geq t_0 \end{array} \right\} y_{zs}(t) \quad t \geq t_0$$

- Zero input response: it tells us what the output is when there is a zero input applied to an initial state.

$$\left. \begin{array}{l} x_1(t_0) \\ u_1(t) = 0 \quad t \geq t_0 \end{array} \right\} y_{zi}(t) \quad t \geq t_0$$

- Every time a signal is applied to a state, an output that is a combination of zero state response and zero input response is observed.

$$\left. \begin{array}{l} x_1(t_0) + 0 \\ 0 + u(t) \end{array} \right\} y(t) = y_{zs}(t) + y_{zi}(t)$$

#### 4.6 Writing a System in a State Space Form Requires:

Step 1: Identify the energy storing elements. This will give us the number of integrators in our model. The number of integrators determines the order of the system.

Step 2: Identify the variables that will be used for modeling. Any system's response is dependent on the input given to the system and also on the integrator output values. The integrator output is denoted  $x$  and the integrator's input is denoted by  $u$ . We need to identify all the  $u$ 's and  $x$ 's.

Step 3: Starting with dynamic elements to obtain the differential equations. The number of differential equations equals the order of the system or the number of energy-storing elements.

Step 4: Write the information obtained using the previous steps into equations using the following form:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

## 4.7 State Space Model Examples

### 4.7.1 Example

In this example as shown in Figure 4.6, the equations are applied to get the state space model.

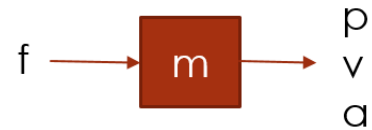


Figure 4.6: A mass pushed by  $f$ . The outputs are  $p$ ,  $v$  and  $a$ .

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

There is a mass pushed by applying force  $f$  on it and the result can be position  $p$ , velocity  $v$  or acceleration  $a$ .

$$f = ma, \quad a = \dot{v}, \quad v = \dot{p}$$

- In this example:  $u = f, \quad y = p$

We select  $p$  and  $v$  to create the state space model equation.

$$\dot{p} = v, \quad \dot{v} = a = \frac{f}{m}$$

$$\begin{bmatrix} \dot{p} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} f$$

$$[p] = [1 \ 0] \begin{bmatrix} p \\ v \end{bmatrix} + [0]f$$

- What if  $u=f$  and  $y = v$ ?

$$f = ma, \quad a = \dot{v} = \frac{f}{m}$$

$$\dot{v} = [0]v + [1/m]f$$

$$v = [1]v + [0]f$$

- What if  $u=f$  and  $y=a$ ?  $a=f/m$  this system doesn't have a state space formulation.

#### 4.7.2 Example

It is required to find the state space model for the circuit from Figure 4.7.

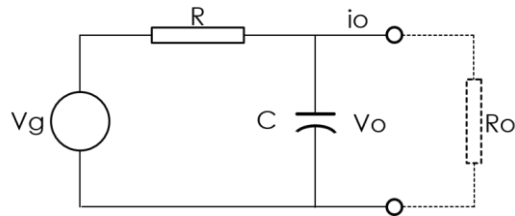


Figure 4.7: Simple RC circuit.

The steps from 4.6 section are applied here.

1. The number of energy storing elements = 1, It is a first order system.
2. The modeling variables are state variables

$$x = V_c = V_o, \quad u = V_g, \text{ as well as parameters } R \text{ and } C.$$

3. Number of differential equations = 1.

$$V_c = \frac{1}{c} \int i_c dt, \quad i_c = i_R - i_o = \left( \frac{V_g - V_c}{R} \right) - \frac{V_o}{R_o}$$

$$\frac{dV_c}{dt} = \frac{[V_g - V_c]}{CR} - \frac{1}{CR_o} \cdot V_c$$

4.  $\dot{x} = Ax + Bu$

$$\dot{V}_c = \left( -\frac{1}{RC} - \frac{1}{R_o C} \right) V_c + \left( \frac{1}{RC} \right) V_g, \quad \text{state equation}$$

$$V_o = [1][V_c] + [0]V_g$$

#### 4.8 Kalman Filter State Model

Simply, Kalman Filter represents the problem mathematically using state equations. Modeling the system mathematically and establishing the system model is a difficult task. Fortunately, Kalman Filter is very popular, so we can find many practical system models to learn how to derive the state model for the Kalman Filter in our type of robot. However, if no reference or no system model examples for similar system can be found, the reader is advised to go deeper in the state space modeling in order to correctly derive his model from the first principle.

Kalman Filter deals with equations as follows:

$$x_{k+1} = Ax_k + w_k$$

$$z_k = Hx_k + v_k$$

$x_k$ : State variable ( $n \times 1$ ).

$z_k$ : Measurement ( $m \times 1$ ).

$A$ : State transition matrix ( $n \times n$ ).

$H$ : State-to-measurement matrix ( $m \times n$ ).

$w_k$ : State transition noise ( $n \times 1$ ).

$v_k$ : Measurement noise ( $m \times 1$ ).

$w_k$  is the noise that affects the system space model and  $v_k$  is the noise measured by the sensor. Matrix  $A$  describes how the system changes along time. It contains the equation motion of the system. The other matrix  $H$  shows the relationship between the measurement and the state variable. It defines how the measurement is mapped to the state variable.

#### **4.9 Covariance Matrix**

There is no method to calculate noise or predict it, but it could be estimated statistically. The noise is distributed normally with a mean of zero. Matrix  $Q$  is the covariance matrix of  $w_k$  (state transition noise) ( $n \times n$ ) and it is a diagonal matrix. While matrix  $R$  is the covariance matrix of  $v_k$  (measurement noise) ( $m \times m$ ) and it is a diagonal matrix. If there is  $n$  noises  $w_1, w_2, \dots, w_n$  and the variance of each variable is  $\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2$  then the covariance matrix will be:

$$Q = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n^2 \end{bmatrix}$$

$Q$  and  $R$  matrices are the design factor of Kalman Filter (Table 4.2). Additionally, they are crucial to the design of Kalman Filter. Appropriate values of  $Q$  and  $R$  should be based on calibration through “trial and error” while maximizing the use of the information on the noise.

$Q$	Error covariance of state transition noise.
$R$	Error covariance of sensor measurements.

Table 4.2:  $Q$  and  $R$  matrices.

#### 4.10 The Effect of Matrix $R$ and $Q$

Using the equations below to find the effect of changing matrix  $R$  or  $Q$ .

$$K = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

$$K = \frac{P_k^- H^T}{(H P_k^- H^T + R)} \text{ (Equation 4.4)}$$

Increasing  $R$  will decrease  $K$ . This will affect the estimate as:

$$\hat{x}_k = \hat{x}_k^- (I - K_k) + K_k z_k$$

When  $K$  decreases, the dependency on the measurement decreases. While when  $K$  increases, the dependency on prediction decreases. On the other hand, increasing  $R$  decreases  $K$  which gives us a steady output less affected by measurements.



$$P_{k+1}^- = AP_k A^T + Q$$

The prediction of error covariance ( $P_k^-$ ) increases if  $Q$  increases. When  $P_k^-$  increases,  $K$  increases. In summary, when  $K$  increases it leads to more influence of measurement. While  $Q$  has opposite influence of matrix  $R$ .

## **4.11 Kalman Filter Examples**

In this section, many examples will be presented. These examples will show how to apply Kalman Filter. Some applications will be easy but sometimes it can be very complicated to create a state space model for the chosen system.

### **4.11.1 One Dimension Kalman Example**

Kalman filter is needed to be used to refine the measurements coming from two sonars in the right or left side of the MCECS-Bot's base. The scenario is that the robot is moving beside a wall and there are two sonars measuring the distance from the wall (Shown in Figure 4.8). There is noise in the sonar measurements (sometimes called acoustic noise) that coming from motors as the magnetic field affects the connections and wires that lead to the sonars. The signals coming from the sonars are very small which makes them very sensitive to any noise. In this simple example, the distances measured by these two sonars will be averaged. Then, Kalman filter will use the averaged value to find the best estimate.

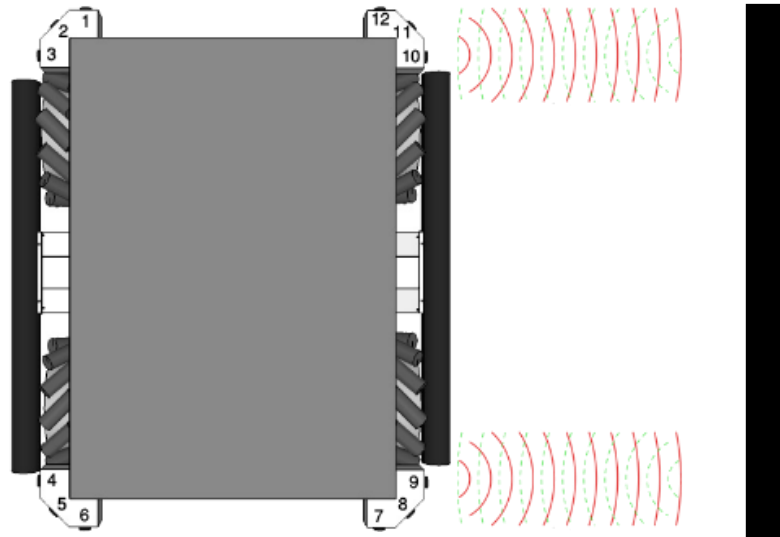


Figure 4.8: Robot following the right wall. The two sonars on the right detect the wall but the distance is not very accurate. In this case, Kalman filter is used to refine the measurements.

**System model:** (in Kalman Filter we just represent the state-state and input-state modelling. No state-output or input-output modelling)

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{w}_k (= \mathbf{0}) \quad \text{Equation 4.5}$$

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \quad \text{Equation 4.6}$$

$$\mathbf{x}_0 = \mathbf{10} \quad \text{Equation 4.7}$$

$$\mathbf{v}_k = N(\mathbf{0}, 2^2) \quad \text{Equation 4.8}$$

The first and second equations (equation 4.5 and equation 4.6) are the state space models. Equation 4.5 shows that there is no change in the state transition as the distance from the wall remains constant. While equation 4.6 maps the state  $\mathbf{x}_k$  of the robot to the measurement  $\mathbf{z}_k$ . As long as there is a noise in the measurement, it is represented in the

second equation as  $v_k$ . The initial distance in inches is shown in the third equation ( $x_0 = 10$ ).  $v_k$  in equation 4 is the measurement noise and it has a normal distribution with mean = 0 and covariance = 4. Error covariance can be known either by trial and error till we find an appropriate value or statistically by taking many measurements and trying to find the normal distribution of them.

From the four previous equations, the following matrices are obtained:

$A = 1$ ,  $H = 1$ ,  $Q = 0$  (There is no noise while transmitting from state to state) and  $R = 4$  (lets assumed that  $R = 4$  came with the sensor taken from the sensor documentation created by the manufacturer)

Finally, an initial values should exist to be fed to the Kalman Filter which are both  $\hat{x}_0^-$  and  $P_0^-$ . If the initial estimates are unknown, a large covariance can be set;  $\hat{x}_0^- = 14$ ,  $P_0^- = 6$ . Figures below show the Matlab codes and the Kalman Filter results. Simply, there is the main function calling three different functions. The first functions is Simple1DKalman (Figure 4.9) function which receives the distances from the sonar simulators and returns the estimated distance. While the second two functions are simulating the sonars measurements which are GetSonar1 and GetSonar2 (Figure 4.10 and Figure 4.11).

Simple1DKalman
<pre> function [D Px K] = Simple1DKalman(AveZ) persistent A H Q R persistent x P persistent firstRun if isempty(firstRun)     A = 1;     H = 1;     Q = 0;     R = 4;     x = 14;     P = 6;     firstRun = 1; end xp = A*x; Pp = A*P*A' + Q; %A' is the transpose matrix for A %and it is the same for H' K = Pp*H'*inv(H*Pp*H' + R);  x = xp + K*(z - H*xp); P = Pp - K*H*Pp; D = x; Px = P; </pre>

Figure 4.9: Simple1DKalman function (it gets the voltage and returns the estimate).

GetSonar1
<pre> function z1 = GetSonar1() w = 0 + 4*randn(1,1); z1 = 35 + w; </pre>

Figure 4.10: GetSonar1 (A function to simulate the first sonar).

GetSonar2
<pre> function z2= GetSonar2() w = 0 + 4*randn(1,1); z2 = 35 + w; </pre>

Figure 4.11: GetSonar2 (A function to simulate the second sonar).

As shown in GetSonar1 and GetSonar2 codes (Figure 4.10 and Figure 4.11), the function simply adds random values to the initial value and sends them as simulated measurements with noise. The main code is shown in Figure 4.12. This code gets the measurements from Sonar1 and Sonar2 and passes them to Kalman Filter algorithm (Simple1DKalman). After getting the estimation, it will show the results. Figure 4.13 shows the result of Kalman Filer that is used in this example.

```
ResultSimple1DKalman
clear all

dt = 0.2;
t = 0:dt:10;

Samples = length(t);

X = zeros(Samples, 1);
Z1 = zeros(Samples, 1);
Z2 = zeros(Samples, 1);
AVEZ = zeros(Samples, 1);
for k=1:Samples
    z1 = GetSonar1();
    z2 = GetSonar2();
    AveZ = (z1+z2)/2;
    D = Simple1DKalman(AveZ);

    X(k) = volt;
    Z1(k) = z1;
    Z2(k) = z2;
    AVEZ(k) = AveZ;
end

figure
plot(t, X, 'o-')
hold on
plot(t, Z1, 'g:*')
plot(t, Z2, 'k:.'
```

```

plot(t, AVEZ, 'r:s')
title('Kalman Filter Estimation');
xlabel('Time(s)');
ylabel('Inche');
grid on
legend('Estimation','Measurements Z1','Measurements Z2','Measurements Average')

```

Figure 4.12: The main code.

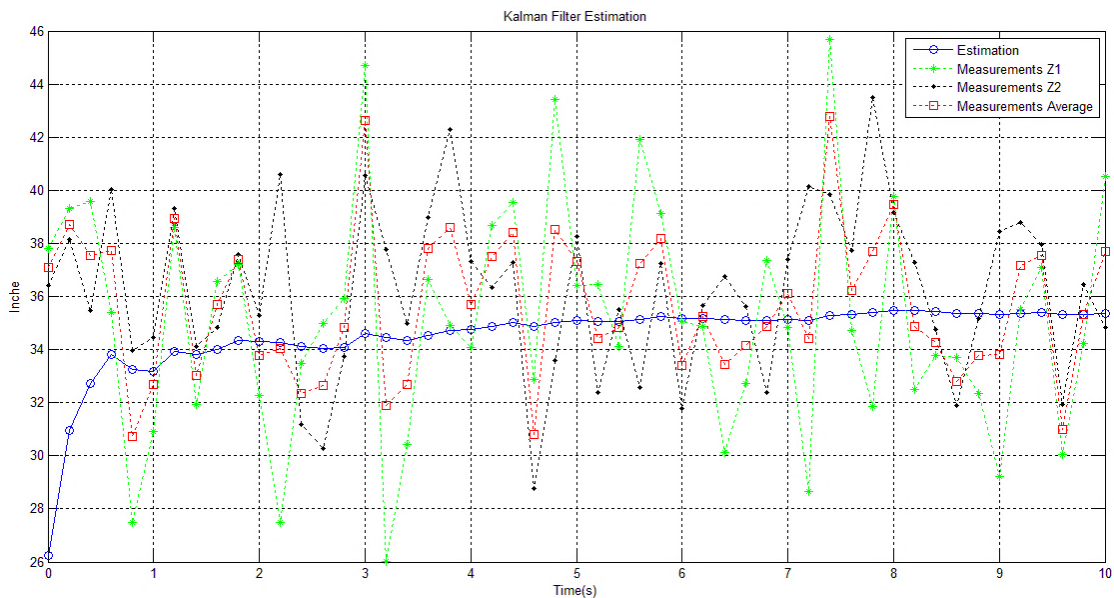


Figure 4.13: Kalman Filter result.

### 4.11.2 Estimating Velocity from Position

So far, Kalman Filter was used to refine noisy signal by getting inaccurate input and producing an estimation as close as possible to the real value. In this example, Kalman Filter will be used to find the best estimation of data that haven't been measured. This will show the unique features of Kalman Filter. Having equations that describe the system will allow Kalman filter estimating data that have not been physically measured. Otherwise,

there is no difference between the Kalman Filter and the Low Pass Filter. Therefore, LPF is unable to find estimate if there are no measurements. This new feature will be explained through the following examples.

### **Another One-Dimensional Kalman Filter Example**

In this example, MCECS-Bot is moving in high speed of 50 inch/s. Kalman filter is used here to find the best estimation of the robot speed. In the first case, the front sonars of the MCECS-Bot will be used to detect a wall by which the robot position will be known. On the other hand, odometry is used to figure out the robot's speed. This example is also considered as it is a one-dimensional Kalman filter example. However, it is different than the first example because we are trying to find another quantity that has not been measured yet. However, the equation of the Kalman and the Matlab code should remain the same. The only difference will be the way that the state space model is formulated. The results in both cases should be both the velocity and the position. In one of the cases, we measure the velocity and estimate the position. In other case, we measure the position and estimate the velocity. If we go back to Low Pass Filter, we will notice that LPF would not be able to find the estimation of the velocity or position if they have not been measured

#### **4.11.2.1 First case**

Velocity and position are required to be estimated from measuring the position:

The state space model for the robot is:

$$x_{k+1} = Ax_k + w_k$$

$$A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, H = [1 \ 0]$$

$\Delta t$  is the time when the measurements were taken.

$$\begin{bmatrix} position \\ velocity \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} position \\ velocity \end{bmatrix}_k + \begin{bmatrix} 0 \\ w_k \end{bmatrix} \quad \text{Equation 4.9}$$

To find whether the above state space model represents the system or not, equation 4.9 will be solved.

$$= \begin{bmatrix} position + velocity \cdot \Delta t \\ velocity + w_k \end{bmatrix}_k$$

$$position_{k+1} = position_k + velocity_k \cdot \Delta t,$$

$$velocity_{k+1} = velocity_k + w_k,$$

These expressions tell us that the current velocity equals the previous velocity plus noise.  $w_k$  is the system noise which comes from friction, error in the engine controller or gears. The system noise can be the gear slippage (there are gears between the encoder and wheels and between wheels and motors). Friction between wheels and ground is another kind of noise. In our robot, the friction is magnified due to the use of Mecanum wheels. Because position is measured, there is no system noise accompanied with position while we transit from state to state.

$$z_k = Hx_k + v_k$$

$$= [1 \ 0] \begin{bmatrix} position \\ velocity \end{bmatrix}_k + v_k \quad \text{(Equation 4.10)}$$



This expression (Equation 4.10) maps the input measurement which is the position to the state variable (position). Here, the noise showed up as there is a noise that accompanies the measurement. So far,  $A$  and  $H$  were found and we proved that the system model exactly describes the robot model in the example.  $Q$  and  $R$  are needed to be found.  $R$  is the covariance matrix of the sensor measurements which is given by the sensor manufacturer. If it is not given the value should be found statistically or by experimenting with different values that can be close to the real one. The sonar measurements are taken from the front the sonars of the MCECS-Bot. The current data were noisy and were affected by acoustic noise from other sonars. Nevertheless, Kalman Filter did a good job with it as shown in Figure 4.19. This was before adding the power delay circuit and chaining method to the twelve sonars which are explained in more details in the fifth chapter. The modelling of  $w_k$  is more difficult to find. It is also based on experiments and experience.

For the current example:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, R = 50$$

Different values of  $Q$  and  $R$  can be used to appreciate how they affect the results of Kalman Filter. Figure 4.14, 4.15 and 4.16 show the Matlab codes that used in this example.

```

KalmanEstimate
function [pos, vel] = KalmanEstimate(z)
%
%
persistent A H Q R
persistent x P
persistent firstRun

if isempty(firstRun)
    firstRun = 1;

    dt = 0.1;

    A = [ 1 dt;
          0 1 ];
    H = [1 0];

    Q = [ 1 0;
          0 1 ];
    R = 50;

    x = [ 0 10 ]';
    P = 5*eye(2); % create an identity 2x2 matrix
    %and multiplied by 5.

    xp = A*x;
    Pp = A*P*A' + Q;

    K = Pp*H'*inv(H*Pp*H' + R);

    x = xp + K*(z - H*xp);
    P = Pp - K*H*Pp;

    pos = x(1);
    vel = x(2);

```

Figure 4.14: Kalman Filter algorithm code. This code gets the sonar data from the main code. Then, it computes the position and velocity and returns them to the main code to be shown.

```

GetMaxSonar
function h = GetMaxSonar()
%
%
persistent son    % SonartAlt.mat
persistent k firstRun

if isempty(firstRun)
    load Son
    k = 1;

    firstRun = 1;
end

h = son(k);

k = k + 1;

```

Figure 4.15: This code reads a file that contains a real sonar data. It sends the data to the main code.

```

Result1DKalmanChangingPosition
clear all
Nsamples = 250;
Xsaved = zeros(Nsamples, 2);
Zsaved = zeros(Nsamples, 1);

for k=1:Nsamples
    z = GetMaxSonar();
    [pos, vel] = KalmanEstimate(z);

    Xsaved(k, :) = [pos vel];
    Zsaved(k) = z;
end

dt = 1;
t = 0:dt:Nsamples-dt;

figure
hold on
plot(t, Zsaved(:, 'r'))
plot(t, Xsaved(:, 1))
xlabel('Time(s)');
ylabel('Position(m)');

```

```

title('Position Estimation');
legend('Measurements', 'Estimation')
grid on

figure
hold on
[A,H1,H2]=plotyy(t, Xsaved(:, 1),t, Xsaved(:, 2),'plot');
set(get(A(1),'Ylabel'),'String','Estimated Distance')
set(get(A(2),'Ylabel'),'String','Estimated Velocity')
xlabel('Time(s)');
title('Velocity Estimation')
set(H2,'color','red');
set(A,{'ycolor'},{'k'})
set(H1,'LineStyle',':')

legend('Estimated Distance','Estimated Velocity')
grid on

```

Figure 4.16: The main code that gets the measurements from GetMaxSonar and passes them to the Kalman Filter algorithm code. After getting the estimated position and velocity, it will show the results.

The results of the main code (4.16) are shown in Figure 4.17 and Figure 4.28. The figures show that even when the measurement was very bad, Kalman Filter has the ability to refine it and find the closest and clearest signal as much as possible.

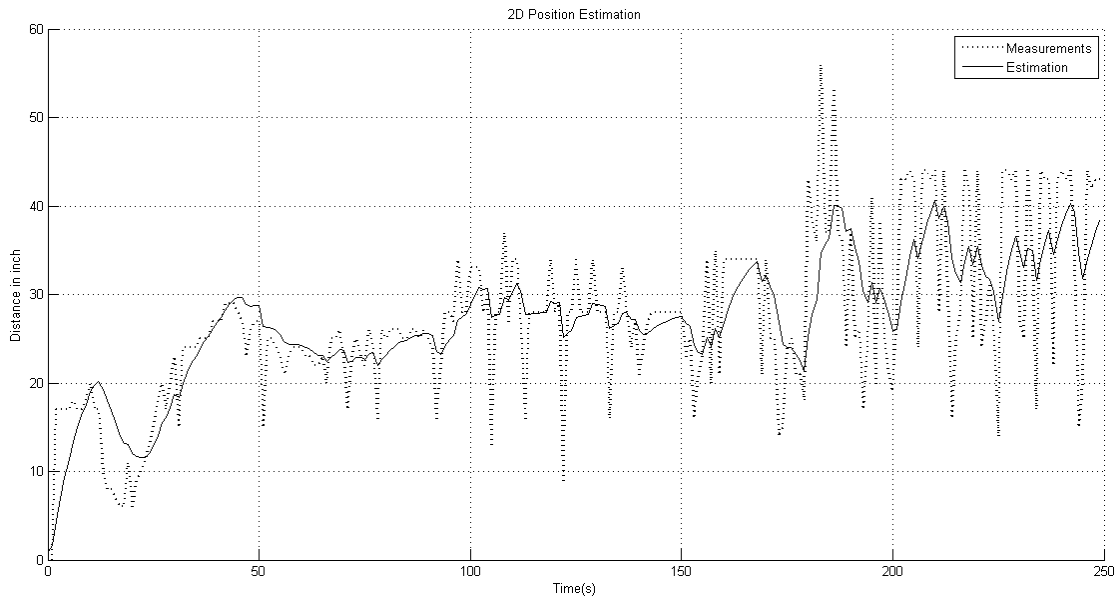


Figure 4.17: Position estimation.

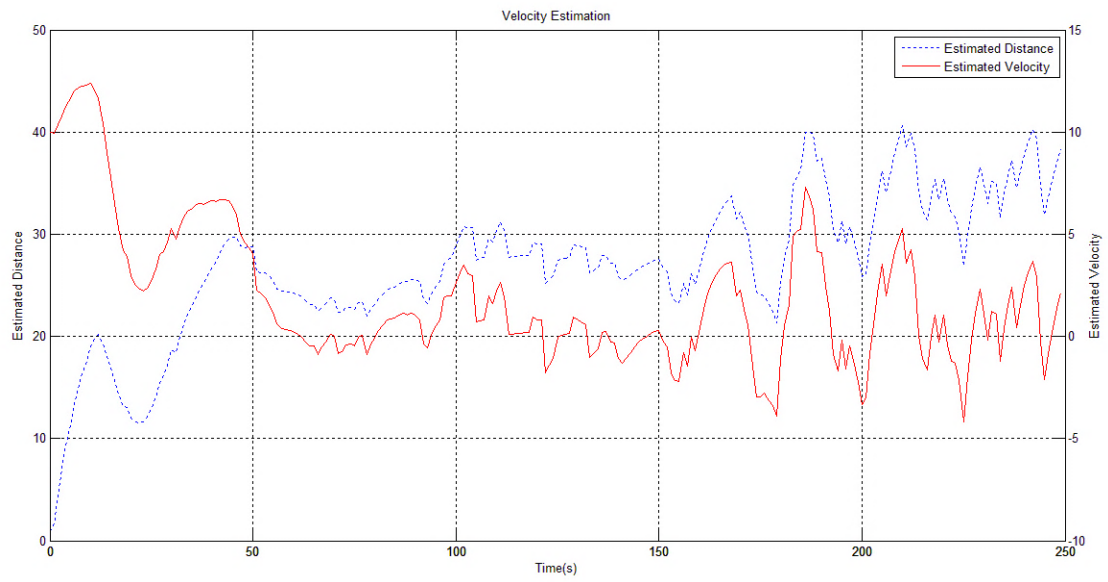


Figure 4.18: Velocity estimation.

#### 4.11.2.2 Second case

Velocity and position are required to be estimated from measuring the velocity. As stated before, odometry is used here to get the robot's speed (the odometry is simulated here not a real data from the MCECS-Bot):

There is nothing that will be changed here except for the measurement transition equation in the state space model equation.  $H$  will become  $[0 \ 1]$  instead of  $[1 \ 0]$  as the input (or measured data) is mapped to the velocity and not to the position state variable.

$$\begin{aligned} z_k &= Hx_k + v_k \\ &= [0 \ 1] \begin{bmatrix} position \\ velocity \end{bmatrix}_k + v_k \end{aligned}$$

The code will be slightly changed as shown in Figure 4.21.

The Matlab code of Kalman Filter is shown in Figure 4.19. `GetVelocity()` (Figure 4.20) is a function that is used to simulate the robot speed. The main code is shown in Figure 4.21 and it gets the velocity measurement from `GetVelocity()` and passes it to `KalmanVelEstimate()` code. Then, it shows the result of the estimations.

```

KalmanVelEstimate
function [pos, vel] = KalmanSpeedEstimate(z)
persistent x P A H Q R firstRun
if isempty(firstRun)
    firstRun = 1;
    dt = 0.1;
    A = [ 1 dt;
          0 1 ];
    H = [0 1];
    Q = [ 1 0;
          0 2 ];
    R = 12;
    x = [ 0 0.3 ]';
    P = 5*eye(2);
end
xp = A*x;
Pp = A*P*A' + Q;
K = Pp*H'*inv(H*Pp*H' + R);
x = xp + K*(z - H*xp);
P = Pp - K*H*Pp;
pos = x(1);
vel = x(2);

```

Figure 4.19: Kalman Filter algorithm code. It receives the sonar data from the main code and sends the estimation back to the main code to be presented.

```

GetVelocity
function CurrentVel = GetVelocity()
persistent PreVel PrePos
if isempty(PrePos)
    PrePos = 0;
    PreVel = 50;
end
dt = 0.1;
VelErr = 0 + (randn*2);
PrePos = PrePos + PreVel*dt; % true position
PreVel = 50 + VelErr; % true speed
CurrentVel = PreVel;

```

Figure 4.20: The velocity generator code. This code emulates the velocity of the robot.

```
Result1DKalmanPosVel

clear all
dt = 0.1;
t = 0:dt:10;
Samples = length(t);
X = zeros(Samples, 2);
Z = zeros(Samples, 1);

for k=1:Samples
    CurrentVel = GetVelocity();
    [pos, vel] = KalmanVelEstimate(CurrentVel);

    X(k, :) = [pos vel];
    Z(k) = CurrentVel;
end

figure
plot(t, Z(:), 'r','LineStyle',':', 'Marker','.')
hold on
plot(t, X(:, 2))
xlabel('Time(s)');
ylabel('Velocity(m/s)');
title('Estimated Velocity');
legend('Measurements', 'Estimation')
figure
plot(t, X(:, 1))
xlabel('Time(s)');
ylabel('Position(m)');
title('Position Estimation');
legend('Estimated Position')
```

Figure 4.21: The main code that receives the sensor data from GetVelocity() and passes it to Kalman Filter algorithm. Then, it shows the result of the Kalman Filter estimation.

The results of the main code are shown in the Figures below (Figure 4.22 and Figure 4.23):



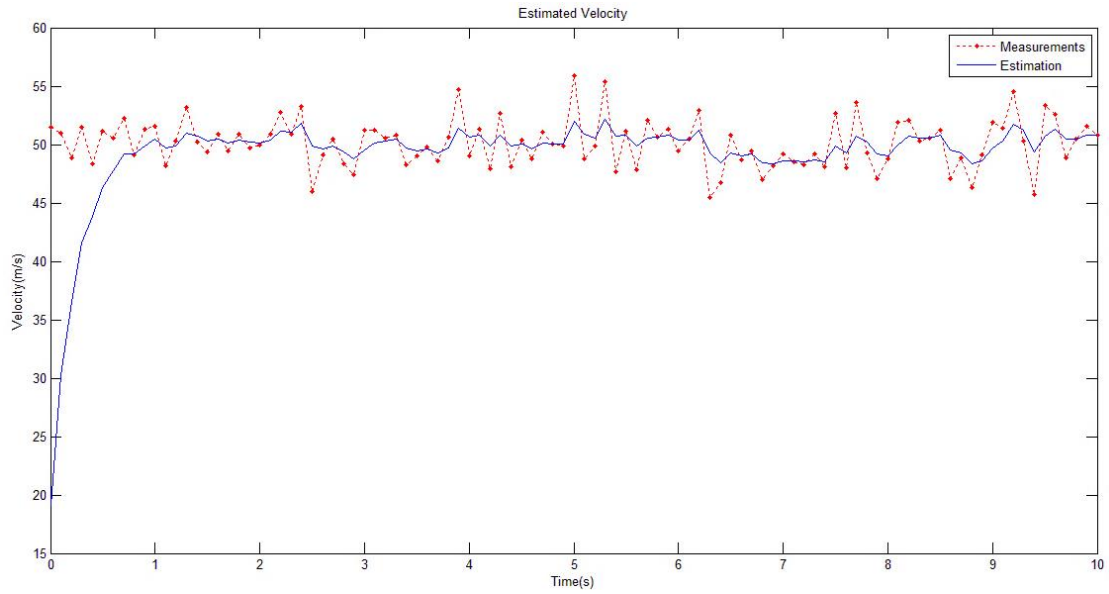


Figure 4.22: Velocity estimation.

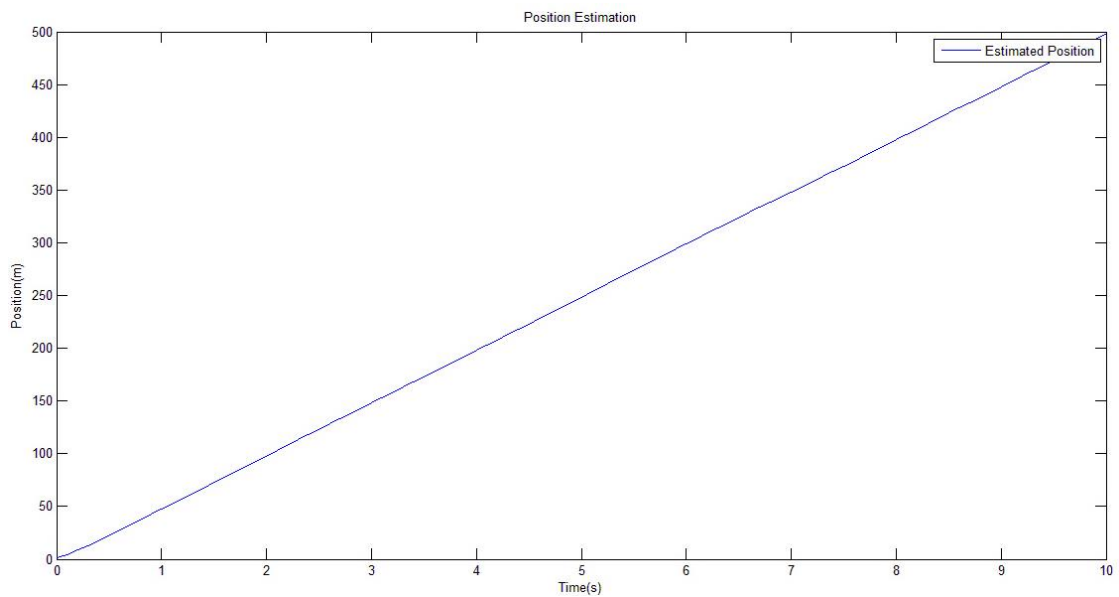


Figure 4.23: Position estimation.

This example showed us how important the state space model is. This importance was obvious when Kalman Filter was able to estimate the position with velocity only (or the opposite) in addition to refining the coming signal. On the other hand, LPF adds the

current measurement (multiplied with some weight) to the previous estimate (multiplied with some weight too). Basically there is no estimation without measurement. In Kalman Filter there is estimation because the mathematical model of the system is known. The explanation of this ability is that the system model that describes the state variables will allow Kalman Filter to estimate the physical quantity that has not been measured. System model is very important and as much as it is close to the real system it will determine the accuracy of the estimation. If it is not it will go far from the real values and may cause a damage to the system.

### 4.11.3 Two Dimension Kalman Filter

The previous examples were a one-dimensional Kalman Filter as there was one input or one set of data. Therefore, the Gaussian of the estimation will be similar to the one in Figure 4.26. Simply, Kalman Filter gives the estimated value as a normal distribution. The one in the mean is going to come with the highest probability and the signal will depend on the width of the bell.

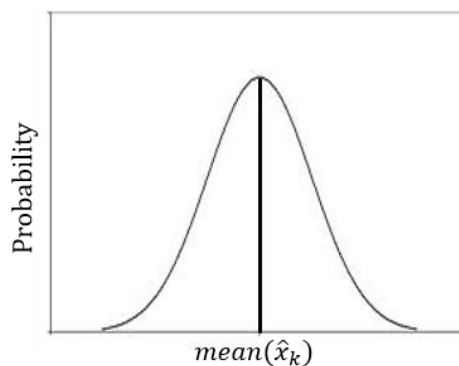


Figure 4.24:  $x_k \sim N(\hat{x}_k, P_k)$ .

Two-dimensional Kalman Filter means that the state of the system has two variables that determine the state of the system. For instance, there are two coordinates; x and y that determine the position of the robot. The Gaussian of the Kalman Filter estimation has a different representation than the one-dimensional from Figure 4.24. To have a clearer understanding, the map in Figure 4.25 is a simple corridor and there is a robot moving there. The two-dimensional Gaussian is shown as a set of circles in the below figure.

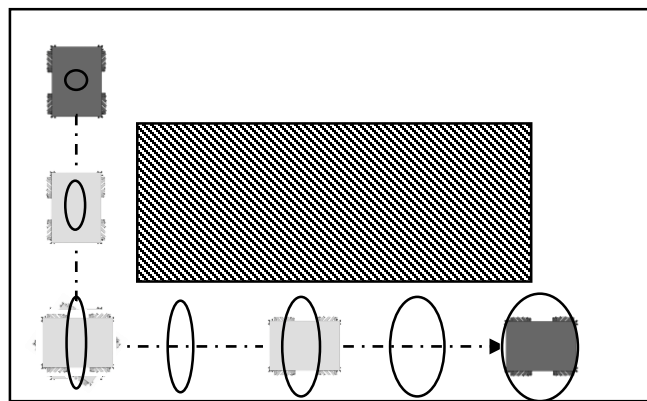


Figure 4.25: Two dimension Gaussian.

The current example is about the MCECS-Bot moving in a room. Based on the sonar measurements from the front and right sonars robot position can be obtained (as illustrated in Figure 4.25). Kalman Filter is used here to optimize the acquired position and estimate the velocity of the robot. The resultant position is presented on x,y coordinates. As shown in the figure above, the Gaussian is represented as ellipse and not in a bell shape. This ellipse shape is like two intersected bells; one is on x-axis and the other one on the y-axis. Their intersection produces an ellipse shape.

#### 4.11.4 Changing Matrices $Q$ and $R$

Here I use the example in the one dimension section to show how the results will be affected by changing  $Q$  or  $R$  values.  $Q$  and  $R$  have opposite effect on the estimation. Increasing  $Q$  will increase the measurement weight. On the other hand, when  $Q$  is decreasing, the estimate will be less affected by measurement and it will converge the previous estimate as well as the result will be smoother. In this example the matrix  $Q$  or  $R$  is multiplied by 0.01. Therefore, the modified code will be only the KalmanEstimate as shown below:

```
Q = 0.01*eye(4);
```

Or

```
R = 1/100 * 50;
```

The result shows how using a smaller  $Q$  affects the result (Figure 4.26).

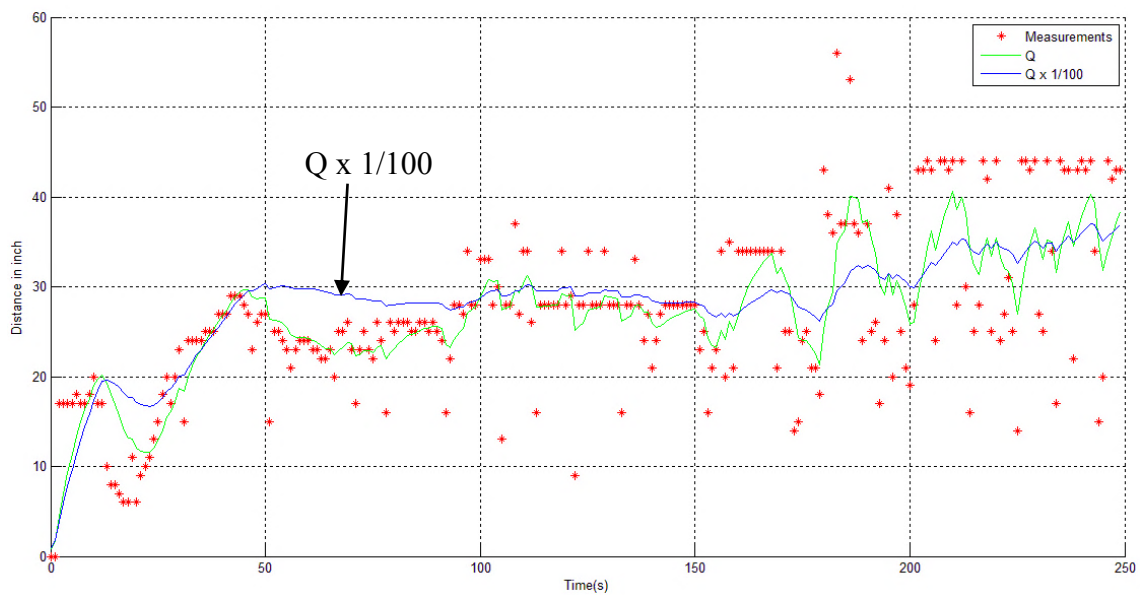


Figure 4.26: 1/100  $Q$ .

It is noticeable that the line that indicated with  $Q/100$  is smoother (Figure 4.26), which means that the estimate was less affected by the measurements and there is more contribution to the previous estimate. Contrary, (Figure 4.27), the opposite happened when the  $R$  reduced 100 times. The indicated line with  $R/100$  were following the stars (measurements), which means that the measurements have more weight on the output estimate.

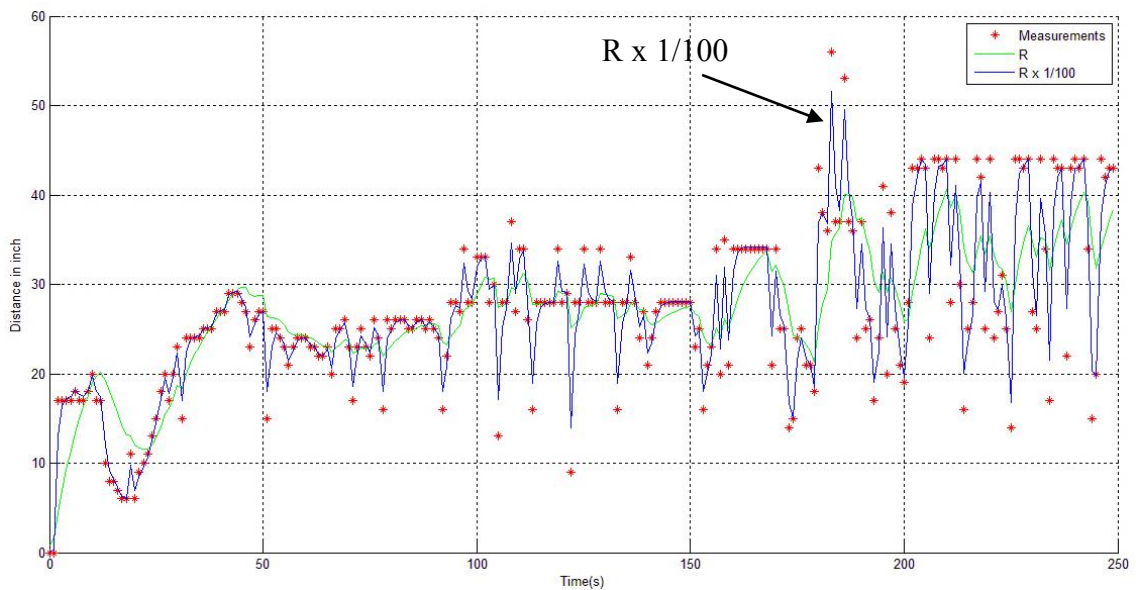


Figure 4.27:  $1/100 R$ .

## 4.12 Sensor Fusion <sup>4</sup>

In many cases, sensors are not accurate devices. Therefore, acquiring accurate data required expensive sensors (The accuracy of sensors are an important matter). However, using different sensors to get a certain data is applicable. The technique of fusing two (or more) different sensors in order to get the best features of each in a one output is called sensor fusion. In this section, an example will show the ability of Kalman Filter to apply this technique. The example is about using a gyroscope and accelerometer to obtain the horizontal altitude of a helicopter.

Gyroscope is used to find the angular velocity on each axis. However, the rate gyros orientation sensors are aligned with the body frame of the sensor, so that if inertial frame data is needed, the sensor outputs must be converted from the body frame to the inertial frame. In other words, Euler angles are the rate of the change in the angles (the alignment doesn't matter). The example Figure 4.28 shows an airplane rotated by Yaw angle. There are new axis different than the original ones. The Euler is about these new axis wherever and however they are obtained but the Gyro is given the angular velocity on each original axis. Hence, a conversion is needed to convert the Angular velocity coming from the gyro to the Euler angles.

---

<sup>4</sup> This section is taken from “Kalman Filter for Beginners” [12]. Except some additional comments that I added to the codes and equations.

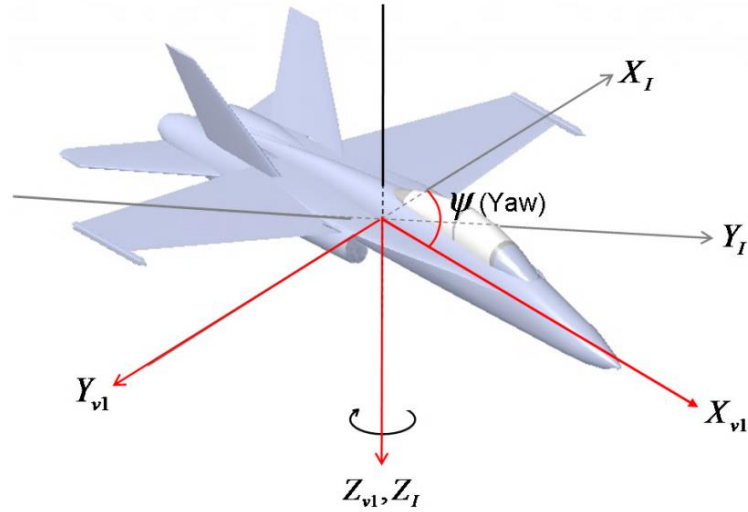


Figure 4.28: The original axes are red and the new axes are gray after Yaw angle rotation [13]

To convert the angular velocities to Euler angles, the below equation is used:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

$p$ ,  $q$  and  $r$  are the angular velocity on each axis.  $\theta$ ,  $\phi$  and  $\psi$  are the Euler angles. Simply, the angular velocity are converted first then the final result is integrated to get each Euler angle.

It is needed to design a Kalman Filter for a helicopter that has both a gyroscope and an accelerometer. Kalman Filter is used here to implement the sensor fusion technique in order to get the horizontal altitude out of these two sensors. A 0.2 Hz sinusoidal signal  $\pm 30$  (as shown in Figure 4.29) is used to test using either sensor or to test them together after having the sensor fusion in Kalman Filter.

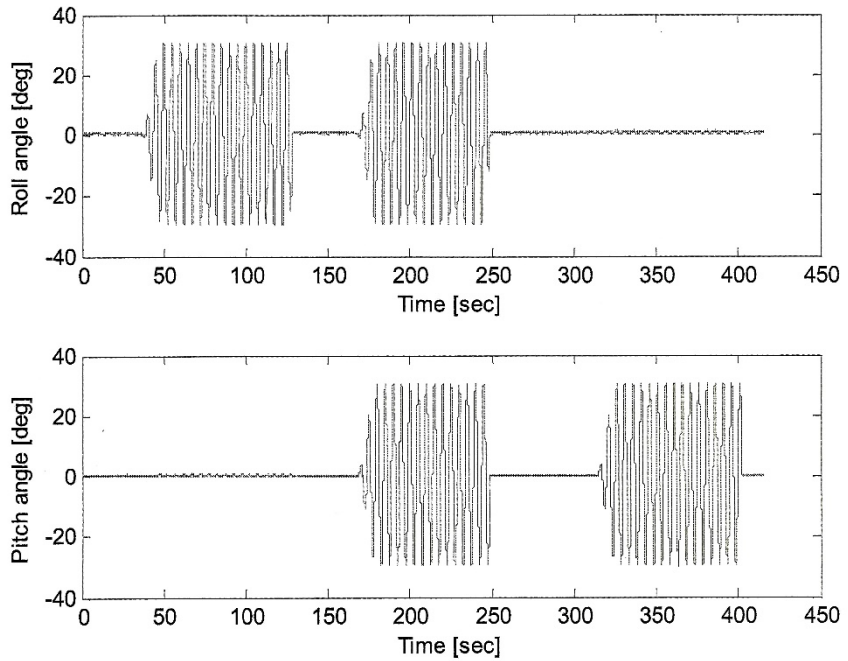


Figure 4.29: Sinusoidal signal used to test gyroscope and accelerometer.

The signal shown in Figure 4.29 will be used to test the gyroscope and the accelerometers for both Roll and Pitch angles response (Figure 4.30 shows these Euler angles on the helicopter).



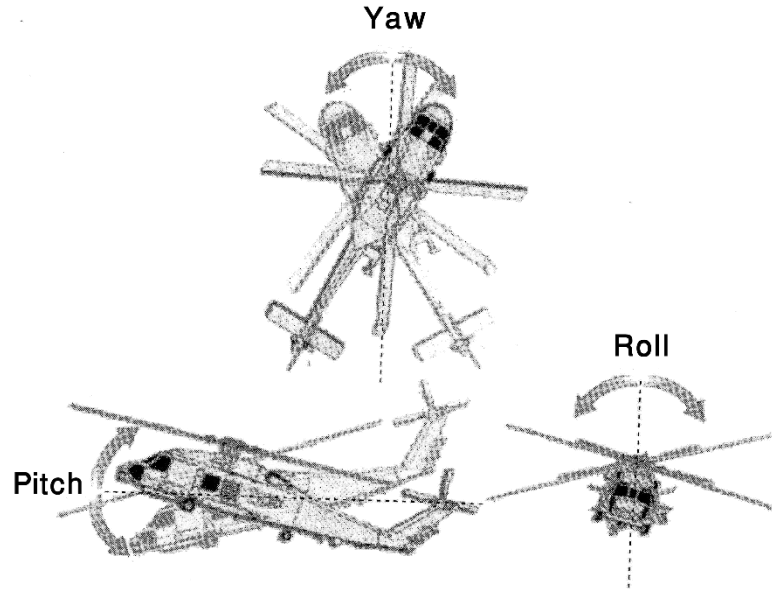


Figure 4.30: Euler Angles.

#### 4.12.1 Using Gyro to get Roll and Pitch angles

In this section, the gyro is used to measure the angular velocities on each axis. The next step is to convert these angular velocities to Euler angles using the following matrix equation:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

The angular velocities that are measured by gyro are showed in Figure 4.31.

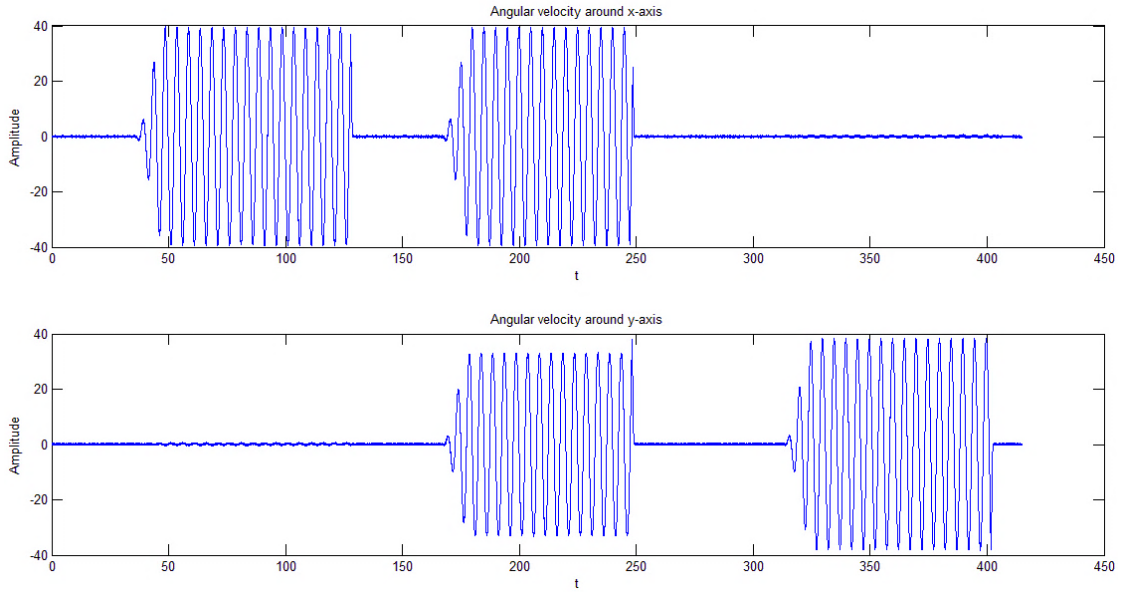


Figure 4.31: Gyros Angular velocities.

Figure 4.32 is the code that simulates the gyroscope measurements. It simulates the angular velocity measurements on each axis and sends them to the main code. Figure 4.33 is the code that converts gyroscope measurements (angular velocities) to Euler angles.

```

GetGyro
function [p q r] = GetGyro()
persistent wx wy wz
persistent k firstRun
if isempty(firstRun)
    load ArsGyro
    k = 1;
    firstRun = 1;
end
p = wx(k);
q = wy(k);
r = wz(k);
k = k + 1;

```

Figure 4.32: GetGyro code.

```

EulerGyro
function [phi theta psi] = EulerGyro(p, q, r, dt)
%
%
persistent prevPhi prevTheta prevPsi

if isempty(prevPhi)
    prevPhi = 0;
    prevTheta = 0;
    prevPsi = 0;
end

sinPhi = sin(prevPhi); cosPhi = cos(prevPhi);
cosTheta = cos(prevTheta); tanTheta = tan(prevTheta);

phi = prevPhi + dt*( p + q*sinPhi*tanTheta +
r*cosPhi*tanTheta );
theta = prevTheta + dt*( q*cosPhi - r*sinPhi );
psi = prevPsi + dt*( q*sinPhi/cosTheta +
r*cosPhi/cosTheta );

prevPhi = phi;
prevTheta = theta;
prevPsi = psi;

```

Figure 4.33: Convert gyro measurements to Euler angles.

Figure 4.34 is the main code that gets the measurements from GetGyro() and passes them to EulerGyro(). Eventually, it presents the results on graphs, which are Figure 4.35 and Figure 4.36.

```

TestEulerGyro
clear all

Nsamples = 41500;
Insamples = 41500;
EulerSaved = zeros(Nsamples, 3);
Input = zeros (Insamples,3);
dt = 0.01;

```

```

for k = 1:Nsamples
    [p q r] = GetGyro();
    [phi theta psi] = EulerGyro(p, q, r, dt);

    Input(k, :) = [p q r];
    EulerSaved(k, :) = [ phi theta psi ];
end
PhiSaved = EulerSaved(:, 1) * 180/pi;
ThetaSaved = EulerSaved(:, 2) * 180/pi;
PsiSaved = EulerSaved(:, 3) * 180/pi;

x = Input(:, 1) * 180/pi;
y = Input(:, 2) * 180/pi;
t = 0:dt:Nsamples*dt-dt;
Figure
subplot(2,1,1)
plot(t, x)
title('Angular velocity around x-axis');
xlabel('t')
ylabel('Amplitude')
subplot(2,1,2)
plot(t, y)
xlabel('t')
ylabel('Amplitude')
title('Angular velocity around y-axis')
Figure
plot(t, PhiSaved)
xlabel('t')
ylabel('Roll rate (deg/sec)')
Figure
plot(t, ThetaSaved)
xlabel('t')
ylabel('Pitch rate (deg/sec)')

```

Figure 4.34: The main code.

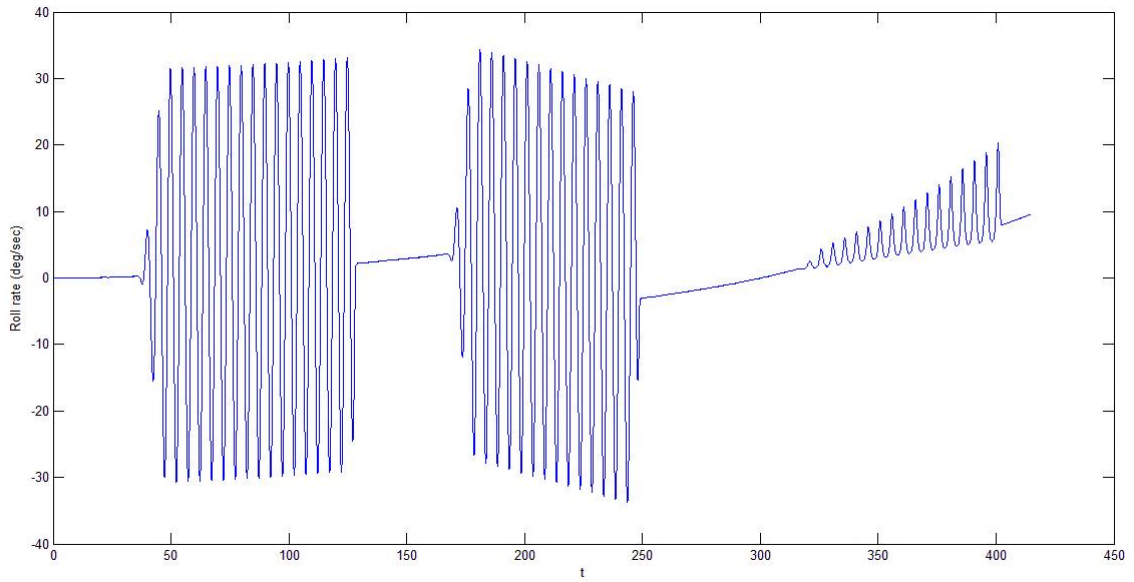


Figure 4.35: Roll angle using gyro.

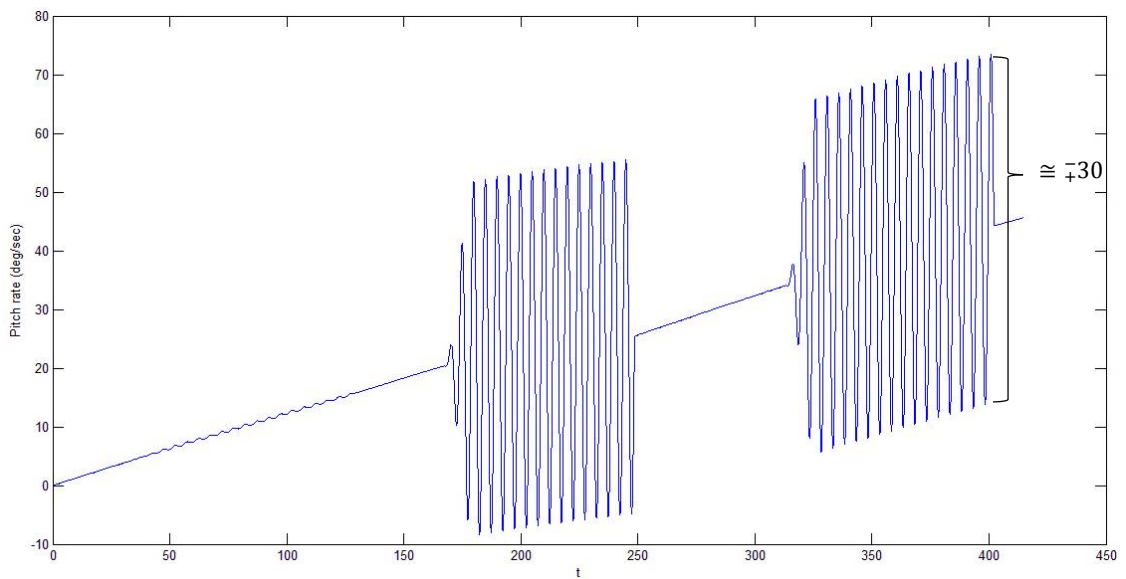


Figure 4.36: Pitch angle using gyro.

The result of using the previous algorithm is shown in the two Figures above. The Roll angle( $\theta$ ) is represented very well (amplitude and duration) except some accumulated

error. On the other hand, there is an accumulated error in the pitch angle( $\theta$ ) and it diverged from the beginning even when there is no oscillation. This is because the influence of the maneuver on the roll axis. The pattern in both roll and pitch angles is represented very well regardless the accumulated error. It was obvious that the amplitude is almost constant.

#### 4.12.2 Using Accelerometers to get Roll and Pitch Angles

In this section accelerometers are used to find the acceleration along each axis and then an equation is used to convert these accelerations to Euler angles. This technique is kind of similar to using the gyro in a sense of using the sensor to do measurements then convert its data to Euler angles. Using the below matrix equation will convert the accelerometer measurements to Euler angles:

$$\begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} 0 & w & -v \\ -w & 0 & u \\ v & -u & 0 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} + g \begin{bmatrix} \sin\theta \\ -\cos\theta\sin\phi \\ -\cos\theta\cos\phi \end{bmatrix} \quad \text{Equation 4.11}$$

Where  $u$ ,  $v$  and  $w$  are the velocities along each axis and  $p$ ,  $q$  and  $r$  are the angular velocities around each axis.  $g$  is the gravitational acceleration.  $f_x$ ,  $f_y$ ,  $f_z$ ,  $p$ ,  $q$ ,  $r$  and  $g$  are known, but  $v$ ,  $u$  and  $w$  are unknown. To know them, expensive sensors should be used. In order to simplify this equation, the previous equation will be considered in both stationary and constant velocity. In the first case, both accelerations ( $\dot{v}$ ,  $\dot{u}$ ,  $\dot{w}$ ) and velocities ( $u$ ,  $v$ ,  $w$ ) along each axis are zero and in the second case  $p$ ,  $q$  and  $r$  are also zero as there is no change in the altitude. The new equation becomes:

$$\begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = g \begin{bmatrix} \sin\theta \\ -\cos\theta\sin\phi \\ -\cos\theta\cos\phi \end{bmatrix}$$

This equation is suitable as there is not that much change in the helicopter velocity. It is a good approximation.

The Euler angles now could be derived:

$$\phi = \sin^{-1}\left(\frac{-f_y}{g\cos\theta}\right)$$

$$\theta = \sin^{-1}\left(\frac{f_x}{g}\right)$$

Figure 4.37 is the code that simulates the measurements of the accelerometer. It simulates the measurements on each axis and sends it to the main code. Figure 4.38 is the code that converts the measurements of the accelerometer (accelerations) to Euler angles.

```

GetAccel
function [ax ay az] = GetAccel()
%
persistent fx fy fz
persistent k firstRun

if isempty(firstRun)
    load ArsAccel
    k = 1;

    firstRun = 1;
end

ax = fx(k);
ay = fy(k);
az = fz(k);

k = k + 1;

```

Figure 4.37: Accelerometer measurements simulator.

```

EulerAccel
function [phi theta] = EulerAccel(ax, ay)
%
%
g = 9.8;

theta = asin( ax / g );
phi = asin( -ay / (g*cos(theta)) );

```

Figure 4.38: Convert Accelerometers' measurements to Euler angles.

Figure 4.39 is the main code that gets the measurements from accelerometer data simulator and passes them to EulerAccel(). Then, it presents the result of EulerAccel() in graphs, which are Figure 4.40 and Figure 4.41. Figure 4.42 shows the accelerations measured by accelerometer (GetAccel()).

```

TestEulerAccel
clear all

Nsamples = 41500;
Insamples = 41500;

EulerSaved = zeros(Nsamples, 2);
Input = zeros (Insamples,2);

for k = 1:Nsamples
    [ax ay] = GetAccel();
    [phi theta] = EulerAccel(ax, ay);

    Input(k, :) = [ax ay];
    EulerSaved(k, :) = [phi theta];
end

PhiSaved = EulerSaved(:, 1) * 180/pi;
ThetaSaved = EulerSaved(:, 2) * 180/pi;

dt = 0.01;

```



```

t = 0:dt:Nsamples*dt-dt;

Figure
subplot(2,1,1)
plot(t, Input(:, 1))
title('Acceleration along x-axis');
ylabel('fx(m/s^2)')
xlabel('t')
subplot(2,1,2)
plot(t, Input(:, 2))
title('Acceleration along y-axis')
ylabel('fy(m/s^2)')
xlabel('t')
Figure
plot(t, PhiSaved)
ylabel('Roll angle(deg)')
xlabel('t')

Figure
plot(t, ThetaSaved)
ylabel('Pitch angle(deg)')
xlabel('t')

```

Figure 4.39: The main code.

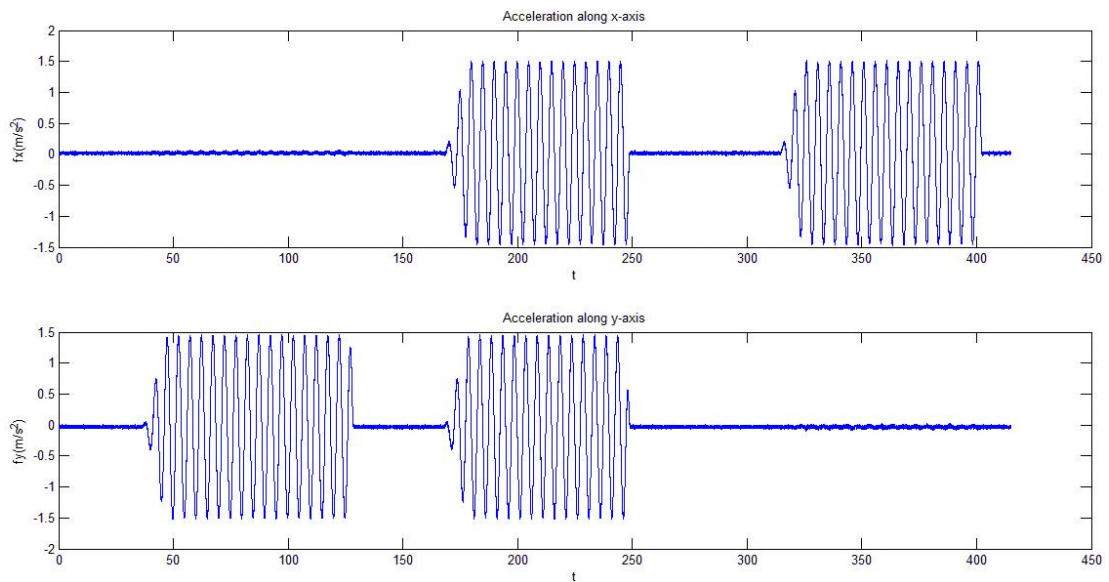


Figure 4.40: Angular velocities measured by Accelerometers.

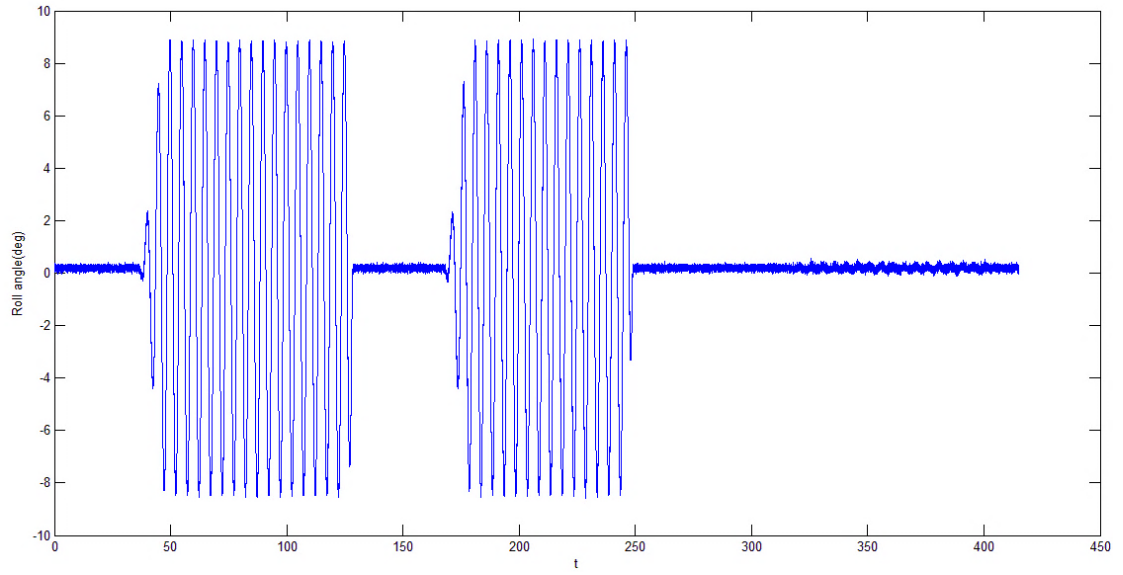


Figure 4.41: Roll angle using the accelerometer. Equation 4.7 is used to convert the angular velocity to Euler angle.

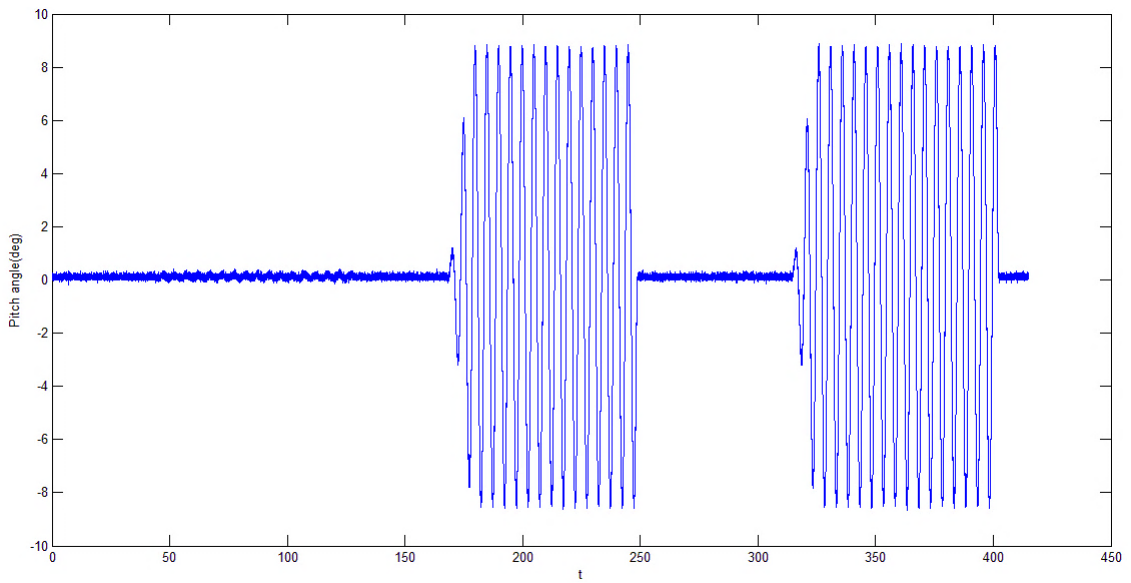


Figure 4.42: Pitch angle using accelerometer. Equation 4.7 is used to convert the angular velocity to Euler angle.

Obviously, the Roll and Pitch angles do not have accumulated error. However, there is a big difference between their amplitude and the one of the original signal. The absence of integration here eliminates the accumulated error.

### 4.12.3 Using Sensor Fusion

It is not useful to use either sensor alone as presented in the previous two sections. Accelerometer has constant error and no divergence. Gyro has got amplitude representation but with divergence. Why don't we use both of them to compensate the shortcoming of each other? The sensor fusion technique is described in the Figure 4.43 below:

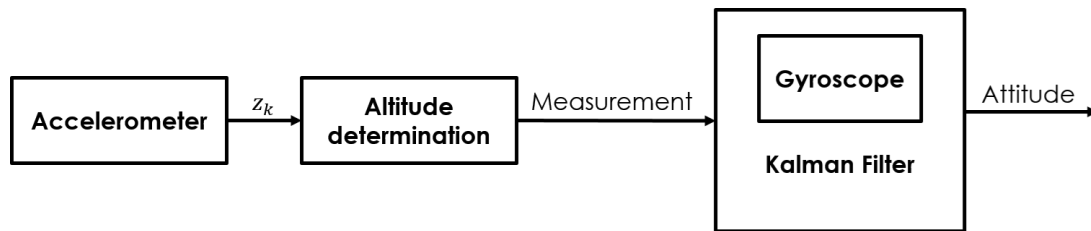


Figure 4.43: Sensor fusion diagram.

#### 4.12.3.1 Altitude-Sensor Fusion-System Model

First of all, the system state variables are:

$$x = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}, \text{ State variables.}$$

The relationship between the angular velocity from gyro and the rate of change in Euler angles is:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi \tan\theta & \cos\phi \tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad \text{Equation 4.12}$$

Equation 4.12 shows the state transition model that has the measurements of gyroscope and the equations that converted them to Euler angles. However, we need an equation that has matrix A and this matrix should not include any state variable (matrix equation represents the system):

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = [A \text{ MATRIX}] \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} + w \quad \text{Equation 4.13}$$

It is very difficult to extract Euler angles from the (Equation 4.13). Therefore, instead of using Euler angles another representation can be used which is the quaternion<sup>5</sup>.

$$x = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}$$

The relationship between the angular velocity from gyro and the quaternion is:

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}$$

---

<sup>5</sup> Quaternion is a method used to rotate a point on three dimensional space. There is no rule to multiply and divide complex numbers on three dimensional space (they are called triples). Nevertheless, the quaternion solved this problem, which took William Rowan Hamilton years of his life to find a three dimensional number systems [54], but he find a solution after he looked at four dimensional space. He discovered the fundamental formula for quaternion multiplication which is:

$$i^2 = j^2 = k^2 = ijk = -1.$$

The system model is:

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}_{k+1} = \overbrace{\left[ I + \Delta t \cdot \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \right]}^A \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}_k$$

The measurements fed to Kalman Filter are coming from the accelerometer. However, we need to convert the Euler angles coming from accelerometer to the quaternion.

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \cos \frac{\phi}{2} \cos \frac{\theta}{2} \cos \frac{\psi}{2} + \sin \frac{\phi}{2} \sin \frac{\theta}{2} \sin \frac{\psi}{2} \\ \sin \frac{\phi}{2} \cos \frac{\theta}{2} \cos \frac{\psi}{2} - \cos \frac{\phi}{2} \sin \frac{\theta}{2} \sin \frac{\psi}{2} \\ \cos \frac{\phi}{2} \sin \frac{\theta}{2} \cos \frac{\psi}{2} + \sin \frac{\phi}{2} \cos \frac{\theta}{2} \sin \frac{\psi}{2} \\ \cos \frac{\phi}{2} \cos \frac{\theta}{2} \sin \frac{\psi}{2} + \sin \frac{\phi}{2} \sin \frac{\theta}{2} \cos \frac{\psi}{2} \end{bmatrix}$$

All the state variables are measured which means:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

So far, the use of sensor fusion was intended to be used. State variable has been changed from Euler angles to quaternion and the measurement has also been changed from Euler angles to quaternion. Now, after having the system model, the next step should be easy which is designing the Kalman Filter.

Q and R are the noise covariance and it is hard to formulate them in a theoretical manner. Therefore, it is necessary to experimentally figure out what is the error distribution of Q and R.

$$Q = \begin{bmatrix} 0.0001 & 0 & 0 & 0 \\ 0 & 0.0001 & 0 & 0 \\ 0 & 0 & 0.0001 & 0 \\ 0 & 0 & 0 & 10.0001 \end{bmatrix}, R = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

The initial values for the state variables and error covariance matrix are:

$$\dot{x}_k^- = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, P_k^- = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The quaternion initialization means that all Euler angles are initialized to zero.

We used the measured Euler angles to make the system matrix  $A$  and the input to Kalman or the measurements will come from the accelerometers. Figure 4.44 shows Kalman Filter algorithm and Figure 4.45 shows the code that converts the Euler angles to quaternion. Figure 4.46 contains the main code that calls other function and shows the results. The last figures (Figure 4.47 and Figure 4.48) show how the accumulated error of the gyroscope and the inaccuracy of the accelerometers have been eliminated. Kalman Filter is able to solve the problem of sensor fusion. In this example, Yaw angle is not required since it has been set to zero.

```

EulerKalman
function [phi theta psi] = EulerKalman(A, z)
persistent H Q R x P firstRun
if isempty(firstRun)
    H = eye(4);
    Q = 0.0001*eye(4);
    R = 10*eye(4);
    x = [1 0 0 0]';
    P = 1*eye(4);
    firstRun = 1;
end
xp = A*x;
Pp = A*P*A' + Q;
K = Pp*H'*inv(H*Pp*H' + R);
x = xp + K*(z - H*xp); % x = [ q1 q2 q3 q4 ]
P = Pp - K*H*Pp;
phi = atan2( 2*(x(3)*x(4) + x(1)*x(2)), 1 - 2*(x(2)^2 + x(3)^2) );
theta = -asin( 2*(x(2)*x(4) - x(1)*x(3)) );
psi = atan2( 2*(x(2)*x(3) + x(1)*x(4)), 1 - 2*(x(3)^2 + x(4)^2) );

```

Figure 4.44: Kalman Filter algorithm.

```

EulerToQuaternion
function z = EulerToQuaternion(phi, theta, psi)
sinPhi = sin(phi/2); cosPhi = cos(phi/2);
sinTheta = sin(theta/2); cosTheta = cos(theta/2);
sinPsi = sin(psi/2); cosPsi = cos(psi/2);

z = [ cosPhi*cosTheta*cosPsi + sinPhi*sinTheta*sinPsi;
      sinPhi*cosTheta*cosPsi - cosPhi*sinTheta*sinPsi;
      cosPhi*sinTheta*cosPsi + sinPhi*cosTheta*sinPsi;
      cosPhi*cosTheta*sinPsi - sinPhi*sinTheta*cosPsi;
      ];

```

Figure 4.45: Convert Euler angles to Quaternion.

```

TestEulerKalman
clear all

Nsamples = 41500;
EulerSaved = zeros(Nsamples, 3);

dt = 0.01;

for k=1:Nsamples
    [p q r] = GetGyro();
    A = eye(4) + dt*1/2*[ 0 -p -q -r;
                        p 0 r -q;
                        q -r 0 p;
                        r q -p 0
                        ];

    [ax ay] = GetAccel();
    [phi theta] = EulerAccel(ax, ay);
    z = EulerToQuaternion(phi, theta, 0);

    [phi theta psi] = EulerKalman(A, z);

    EulerSaved(k, :) = [ phi theta psi ];
end

PhiSaved = EulerSaved(:, 1) * 180/pi;
ThetaSaved = EulerSaved(:, 2) * 180/pi;
PsiSaved = EulerSaved(:, 3) * 180/pi;

t = 0:dt:Nsamples*dt-dt;

Figure
plot(t, PhiSaved)
xlabel('time (sec)')
ylabel('Roll angle (deg)')

Figure
plot(t, ThetaSaved)
xlabel('time (sec)')
ylabel('Pitch angle (deg)')

Figure
plot(t, PsiSaved)

```

Figure 4.46: The main code that tests Kalman Filter algorithm.



The main code uses the same GetGyro() and GetAccel() code in the previous two sections.

It does the following steps:

- Call GetGyro() function to get the gyro angular velocities ( $p$ ,  $q$  and  $r$ ).
- Use the angular velocities coming from gyro to compute  $A$ .
- Get the measurements from accelerometers ( $ax$ ,  $ay$ ).
- Get  $\phi$  and  $\theta$  by converting the accelerometers measurements using EulerAccel() function.
- Call the EulerToQuaternion() function and put the quaternion in  $z$  matrix.
- Pass matrix  $A$  and  $z$  arguments to EulerKalman() function.
- Get the result and plot them.

It is clear that Kalman Filter gets the closest possible estimates to the real values as shown in Figure 4.47 and Figure 4.48. The sensor fusion model brought the best features of the gyroscope and the accelerometer. The sensors compensate the weaknesses of each other.

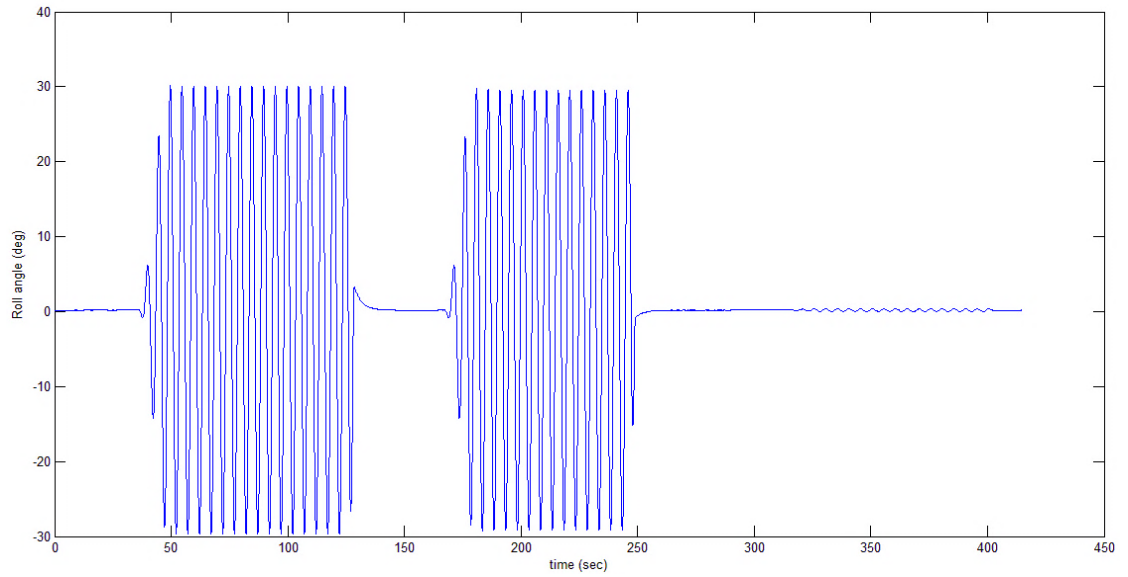


Figure 4.47: Roll angle after using Kalman Filter sensor fusion technique.

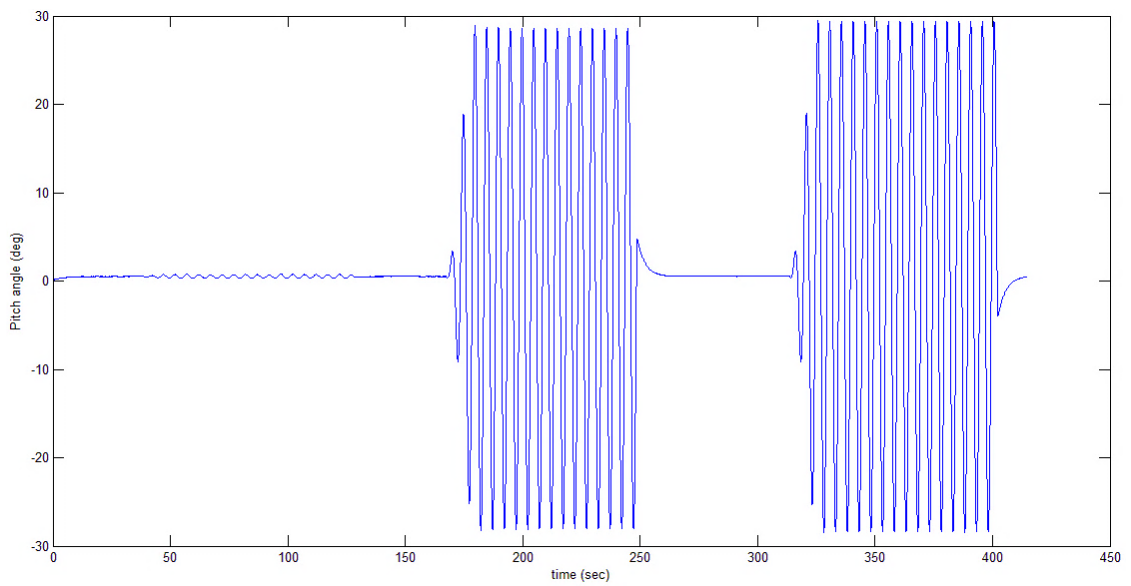


Figure 4.48: angle after using Kalman Filter sensor fusion technique.

### 4.13 Extended Kalman Filter

In previous sections, Kalman filter has been designed to work with linear systems. If the system turns out to be nonlinear, a system linearization is required. Otherwise, an Extended Kalman Filter needs to be designed for this purpose. Extended Kalman Filter is important as most of the systems in the real life are nonlinear. Extended Kalman Filter algorithm is quite similar to Kalman Filter algorithm except two differences as shown in Figure 4.49. Sometimes, it is possible to linearize the system in order to apply Kalman Filter. The previous example that used the quaternion instead of the Euler angles is an example of a linearization process to make the system compatible with Kalman Filter. However, the linearization technique has its limit as it doesn't work for all possibilities. Therefore, a good linearization is required. Otherwise it will diverge.

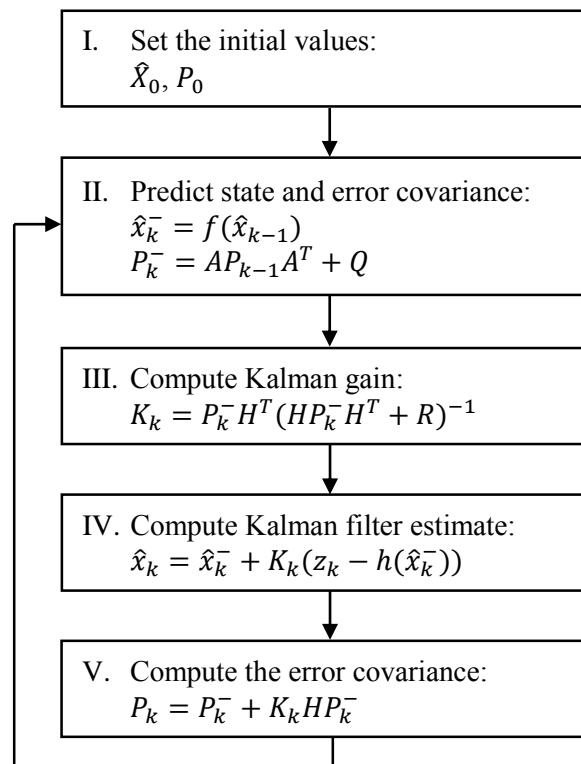


Figure 4.49: Extended Kalman Filter algorithm [12].

As showed in Extended Kalman Filter algorithm, the differences are in the below two equations:

- The second step:

$$\hat{x}_k^- = f(\hat{x}_{k-1})$$

$$\hat{x}_k^- = A\hat{x}_{k-1}$$

- Step three is slightly different as well:

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k))$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k)$$

- The third difference that  $A$  and  $H$  matrices are not obtained directly from the system model equations. Instead, they are obtained using the Jacobian of the nonlinear equations that describes the system.

$$A \equiv \left. \frac{\partial f}{\partial x} \right|_{\hat{x}_k}, \quad H \equiv \left. \frac{\partial h}{\partial x} \right|_{\hat{x}_k^-}$$

Extended Kalman Filter is used with a nonlinear systems, therefore, there are no matrices to map the states or to map the input to state variable. Instead, the current state will be a function of the previous state as well as the current measurement will be a funtion of the current state.

### 4.13.1 Altitude Reference System Example<sup>6</sup>

The same example of the helicopter that I used in the sensor fusion section is used here. The difference is that instead of using quaternion representation instead of the Euler Angles, the system is linearized by figuring out the Jacobian of the equations that describe the system. Extended Kalman Filter is used with the same example in order to compare the results and see how EKF is better than Kalman Filter with nonlinear systems.

Extended Kalman Filter is used to fuse the sensors measurements in order to get the best features of both. Basically, the measurements coming from the gyroscope are used in the state transition equations (equation 4.16). While the measurements coming from the accelerometer are used in the observation equations (equation 4.15).

The measurements are coming from accelerometers:

$$z = \begin{bmatrix} \phi_{Acc} \\ \theta_{Acc} \end{bmatrix} \quad \text{Equation 4.14}$$

where  $\phi_{Acc}$  and  $\theta_{Acc}$  (in equation 4.14) are the Euler angles that came from converting the accelerations measured by the accelerometers.

$$z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} + v \quad \text{Equation 4.15}$$
$$z = Hx + v$$

---

<sup>6</sup> This example is taken from “Kalman Filter for Beginners” [12].

The system variables are:

$$x = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} + w$$

$$\begin{bmatrix} p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \\ p\cos\phi - r\sin\phi \\ q\sin\phi/\cos\theta + r\cos\phi/\cos\theta \end{bmatrix} + w \quad \text{Equation 4.16}$$

$$\equiv f(x) + w$$

The state variable  $x_{k+1}$  in equation 4.16 does not equal a mapping matrix  $A$  multiplied by the state variables  $x_k$ . In other word, it is not possible to extract the  $A$  matrix from the equations on the right side of equation 4.16 because these equations describe a non-linear system. To find  $A$  matrix, the Jacobian for Equation 4.16 must be found:

$$A = \begin{bmatrix} \frac{\partial f_1}{\partial \phi} & \frac{\partial f_1}{\partial \theta} & \frac{\partial f_1}{\partial \psi} \\ \frac{\partial f_2}{\partial \phi} & \frac{\partial f_2}{\partial \theta} & \frac{\partial f_2}{\partial \psi} \\ \frac{\partial f_3}{\partial \phi} & \frac{\partial f_3}{\partial \theta} & \frac{\partial f_3}{\partial \psi} \end{bmatrix}$$

$$= \begin{bmatrix} q.\cos\phi.\tan\theta - r.\sin\phi.\tan\theta & q.\sin\phi.\sec\theta^2 + r.\cos\phi.\sec\theta^2 & 0 \\ -q.\sin\phi - r.\cos\phi & 0 & 0 \\ q.\cos\phi.\sec\theta - r.\sin\phi.\sec\theta & q.\sin\phi.\sec\theta.\tan\theta + r.\cos\phi.\sec\theta.\tan\theta & 0 \end{bmatrix}$$

Below are the figures that contain the code used to simulate the current example.

Figure 4.50 contains the Extended Kalman Filiter algorithm. It receives two vectors and dt

which is the sampling time. The first vector is  $[\phi_a \theta_a]'$  (or  $z$  as it is in EulerEKF code), which has the Euler angles that came from the accelerometer. This vector is used in the observation model. The second vector is  $[p \ q \ r]$  (or rates as it is in EulerEKF code). These are the measurements coming from the gyroscope, which will be used in the state transition model. Figure 5.51 shows the main code that calls different functions. The first two functions are GetGyro() and GetAccel() which are the measurements from the gyroscope and the accelerometer. Second, it passes accelerometer measurements to EulerAccel(ax, ay) to convert the accelerations to Euler angles. Third, it passes the results of EulerAccel(ax, ay) to EKF function (EulerEKF()). Finally, TestEulerEKF gets the results of EulerEKF() and presents them in graphs (Figure 4.52 and Figure 4.53). GetGyro() and GetAccel are not presented here because they were presented in the previous section in the original helicopter example.

EulerEKF
<pre> function [phi theta psi] = EulerEKF(z, rates, dt) persistent H Q R persistent x P persistent firstRun if isempty(firstRun)     H = [ 1 0 0;           0 1 0 ];      Q = [ 0.0001 0 0;           0 0.0001 0;           0 0 0.1 ];      R = [ 6 0;           0 6 ];      x = [0 0 0]';     P = 10*eye(3);      firstRun = 1; </pre>

```

end
A = Ajacob(x, rates, dt);
xp = fx(x, rates, dt);
Pp = A*P*A' + Q;
K = Pp*H'*inv(H*Pp*H' + R);
x = xp + K*(z - H*xp);
P = Pp - K*H*Pp;
phi = x(1);
theta = x(2);
psi = x(3);
%-----
function xp = fx(xhat, rates, dt)
%
%
phi = xhat(1);
theta = xhat(2);
p = rates(1);
q = rates(2);
r = rates(3);
xdot = zeros(3, 1);
xdot(1) = p + q*sin(phi)*tan(theta) +
r*cos(phi)*tan(theta);
xdot(2) = q*cos(phi) - r*sin(phi);
xdot(3) = q*sin(phi)*sec(theta) + r*cos(phi)*sec(theta);
xp = xhat + xdot*dt;
%-----
function A = Ajacob(xhat, rates, dt)
A = zeros(3, 3);
phi = xhat(1);
theta = xhat(2);
p = rates(1);
q = rates(2);
r = rates(3);
A(1,1) = q*cos(phi)*tan(theta) - r*sin(phi)*tan(theta);
A(1,2) = q*sin(phi)*sec(theta)^2 +
r*cos(phi)*sec(theta)^2;
A(1,3) = 0;
A(2,1) = -q*sin(phi) - r*cos(phi);
A(2,2) = 0;
A(2,3) = 0;
A(3,1) = q*cos(phi)*sec(theta) -
r*sin(phi)*sec(theta);
A(3,2) = q*sin(phi)*sec(theta)*tan(theta) +
r*cos(phi)*sec(theta)*tan(theta);

```



```
A(3,3) = 0;
A = eye(3) + A*dt;
```

Figure 4.50: Extended Kalman Filter function that receives arguments and return the estimation.

```
TestEulerEKF
clear all
Nsamples = 41500;
EulerSaved = zeros(Nsamples, 3);
dt = 0.01;
for k=1:Nsamples
    [p q r] = GetGyro();
    [ax ay] = GetAccel();

    [phi_a theta_a] = EulerAccel(ax, ay);

    [phi theta psi] = EulerEKF([phi_a theta_a]', [p q r], dt);

    EulerSaved(k, :) = [ phi theta psi ];
end
PhiSaved = EulerSaved(:, 1) * 180/pi;
ThetaSaved = EulerSaved(:, 2) * 180/pi;
PsiSaved = EulerSaved(:, 3) * 180/pi;
t = 0:dt:Nsamples*dt-dt;
figure
plot(t, PhiSaved)
xlabel('Time(s)');
ylabel('Roll angle(deg)');
figure
plot(t, ThetaSaved)
xlabel('Time(s)');
ylabel('Pitch angle(deg)');
figure
plot(t, PsiSaved)
```

Figure 4.51: The code that test teh EKF and draw the results.

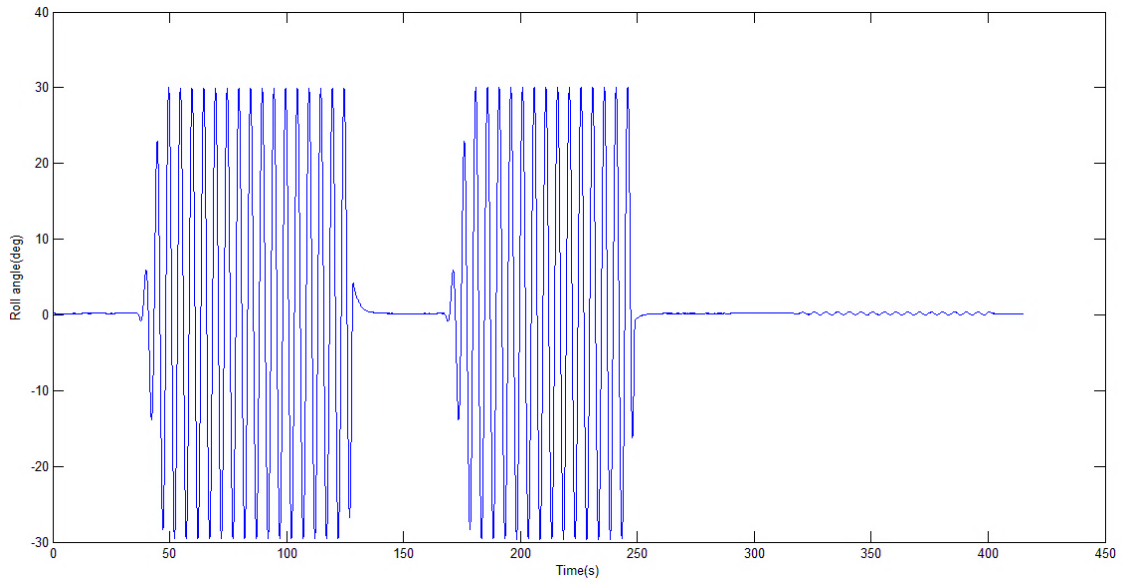


Figure 4.52: Roll angle estimation result of EKF.

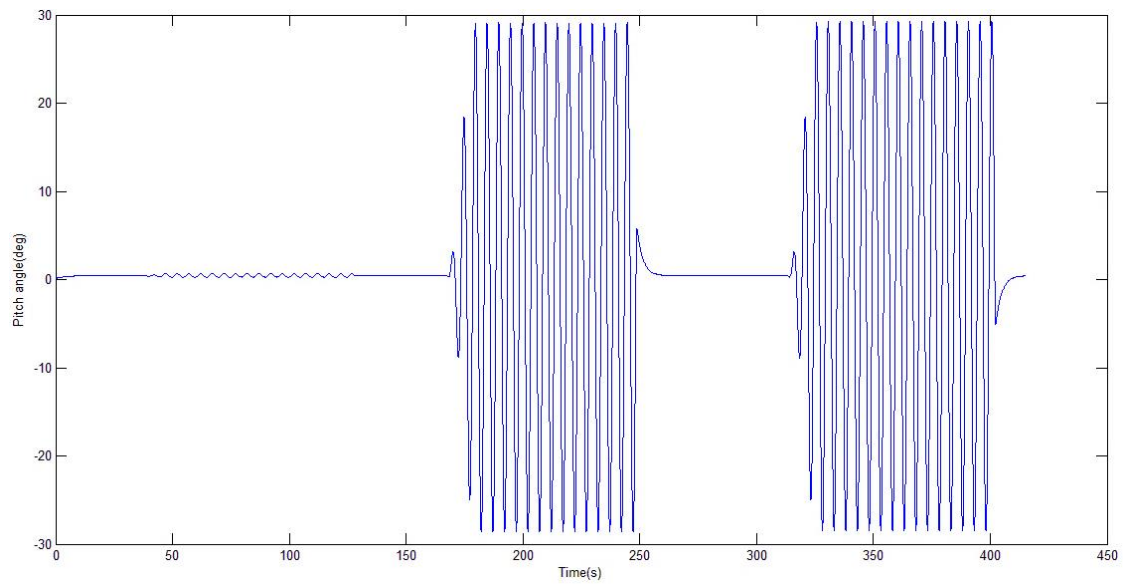


Figure 4.53: Pitch angle estimation result of EKF.

It is obvious that the results in Figure 4.52 and Figure 4.53 are very close to the real ones. Using the quaternoin representation is not always aplicable. In other words, Linear

Kalman Filter is not applicable all the times and sometimes using it without having a good descriptive system equations could have devastating results.

Chapter 5: MCECS Guide Bot - PSU GUIDE ROBOT

## **Chapter 5: MCECS Guide Bot**

MCECS Bot is a robot that can give guided tours inside the PSU FAB Engineering Building while interacting with visitors. The base has four Mecanum wheels which give the base three degrees of freedom. Therefore, MCECS Bot will be able to move forward/backward, left/right and rotate in its position. There are many sensors on the MCECS Bot that provide the main processor (Brain) with different “senses” which allows the robot to do localization, obstacle avoidance and gesture/speech recognition. Since this is a localization thesis, it will focus only on the base sensors which are related to the localization part. The “sense” system of the robot consists of different sensors. There are 12 sonars to detect obstacles, LRF which is a laser range finder used for localization purposes, Kinect used for gesture recognition and localization purposes, encoders used for the odometry and bumpers for emergency stop. There are two ways to interact with the robot which are either voice/gesture recognition or a direct interaction using the attached tablet on the robot. Figure 5.1 shows the MCECS Guide robot diagram. It is possible to control the robot remotely using the Wi-Fi connection.

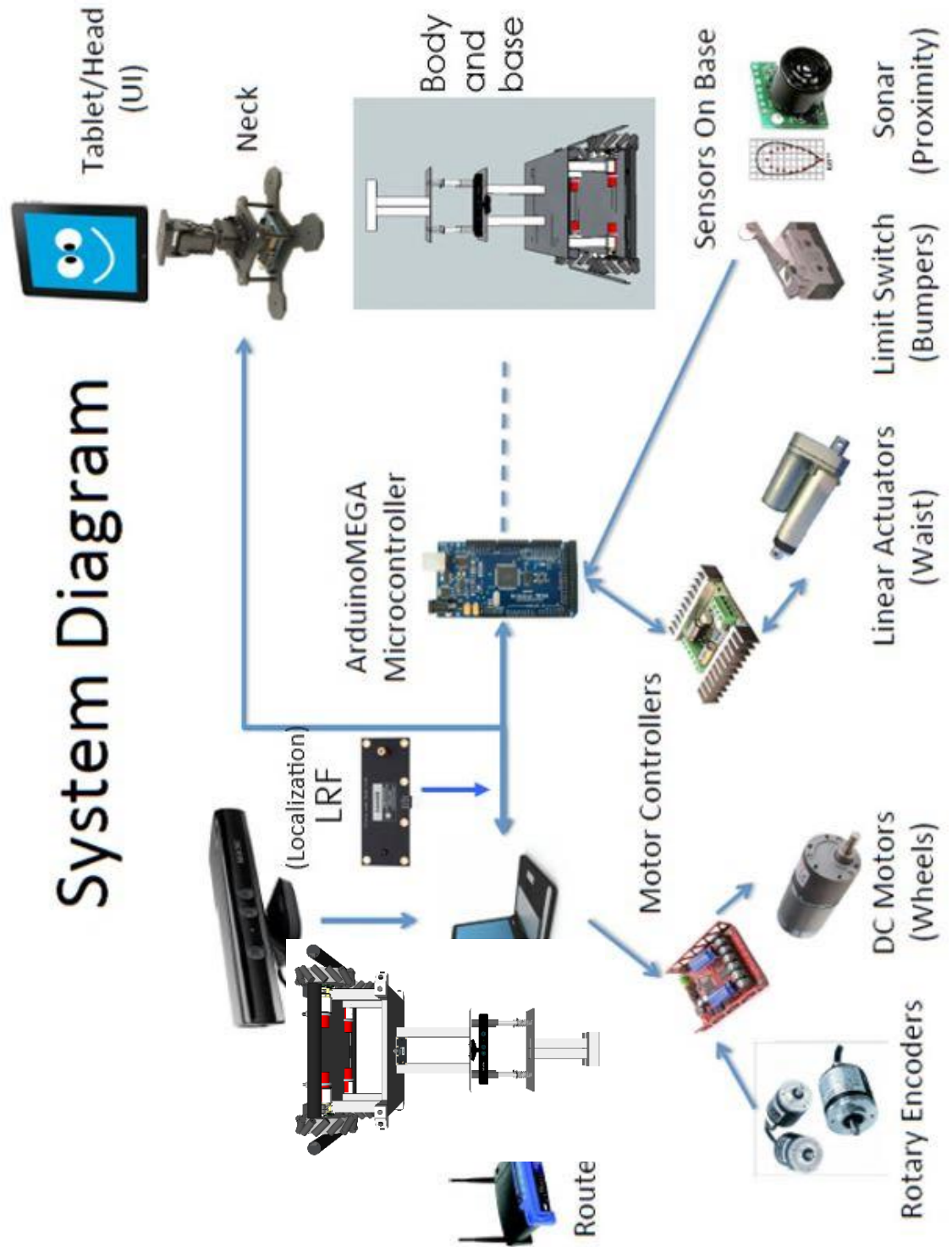


Figure 5.1: MCECS Bot diagram [14].

The idea of building the MCECS Guide Bot came after realizing that PEOPLE-Bot (which is the first guide bot in the robots museum) was not very functional. Professor Perkowski and his students before Fall 2011 were using the PEOPLE-Bot as a platform to improve their localization methods and did experiments (PEOPLE-Bot is shown in Figure 5.2). In Fall 2011, PEOPLE-Bot was improved and lot of issues in the odometry system and the interface circuits were fixed<sup>7</sup>. Additionally, The PEOPLE-Bot team worked on the voice/facial recognition and tested them on the PEOPLE-Bot. Basically, the robot has the map saved in his brain (PC) and when it is given an order to move from a point to another point on the map, the robot was able to achieve it while avoiding obstacles as well. The PEOPLE-Bot is used also to test the software built by the students in order to use them in the main guide robot which is the MCECS Guide Bot.

---

<sup>7</sup> Jim Larson was the team leader in Fall 2011. His team worked on the PEOPLE-Bot to fix the odometry and the interfacing issues.



Figure 5.2: PEOPLE-Bot.

The MCECS-Bot is built from scratch. The project started in Winter 2012 when the first part of the base was built (Figure 5.3) and the first prototype of the neck and the arms were attached. The second step was in Spring 2012 where the second part of the robot's base was finished, the base power and control circuits were built, the battery system was installed, and all other sensors were added to the base (Sonars, encoders and bumpers). Later, since Fall 2012 many teams have worked on the MCECS Guide Bot. Fall 2012 when the right hand and neck were added to the robot. The laser sensor (LRF) is added to improve



the localization functionality of the robot. The software used in the PEOPLE-Bot to implement localization is the MRPT library, which is a collection of C++ libraries created for robot developers. This framework was chosen because guidebot code relies on it as well as it is the most familiar framework to some of the current MCECS bot members [15]). The MRPT library is used again with the MCECS Guide Bot. However, there was an issue as the MRPT library uses Kinect measurements and it was not compatible with the information coming from sonars. Nevertheless, the LRF is used to simulate the Kinect measurements in order to feed them to the MRPT code. Moreover, Sonars can be used to simulate the Kinect measurements to fix the problem of sonars/MRPT compatibility.

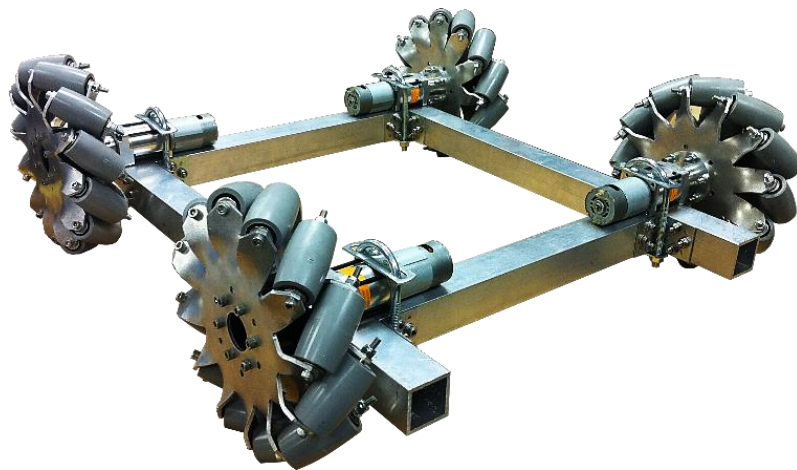


Figure 5.3: The first part of the base.

Eventually, MCECS-Bot is used as part of my thesis as it will be used to apply the Kalman Filter on its navigation part. My thesis is specifically focused on using Kalman Filter for a robot with Mecanum wheels. There are more challenges in using Kalman Filter with Mecanum wheels than using the same technique with traditional wheels. The state variables in this case are the measurements coming from sonars, LRF and odometry.

Therefore, presenting the MCECS Guide Bot and its mechanical design is important to fully understand the Kalman Filter from different perspectives.

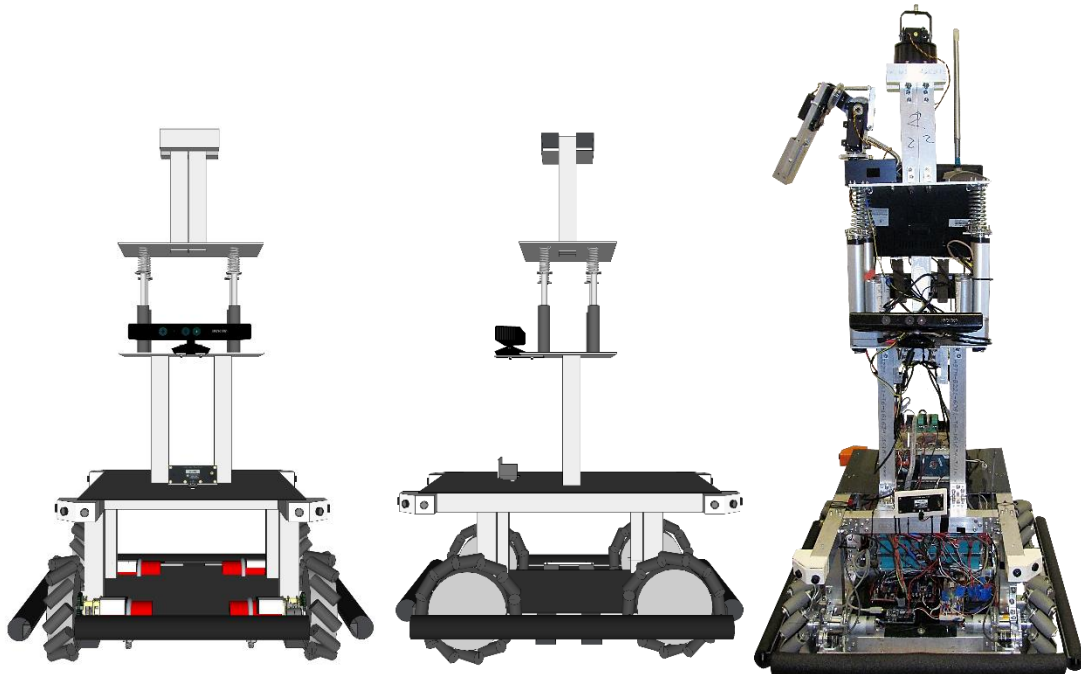


Figure 5.4: PSU Guide Robot (MCECS-Bot).

## 5.1 Base

It is the main part of the robot as it contains the main circuits, sensors and batteries as well as it holds the other robot's body. It is designed to support 160 lbs. The base has 4 Mecanum wheels. Each wheel is run by a model 555 DC motor and gears. There are two motor controllers that control the motors which means each motor controller handles two motors. Sonars, LRF, Batteries, Arduino boards and other management boards are also held by the base. The base is able to move in more degrees of freedom than the traditional robot bases.

### **5.1.1 Mechanical Design and Basic Power Calculations**

In this section, the torque calculation of the DC motors has been covered. This calculation was essential to specify the motor size and type that are suitable for the robot tasks. Then, based on the motor specifications, the gears were chosen. Next, an estimation of the power consumption has been provided based on the DC motor consumption. The motors will be the main power consuming part in the robot. Eventually, after specifying motors, gears and the power consumption, Batteries are chosen to meet the required current by the DC motors and the power consumption by other parts.

#### **5.1.1.1 Torque Calculation**

Figure 5.5 shows a ramp with a certain angle  $\theta$ . The assumption is that the base is moving on this ramp with an angular velocity and acceleration. It is kind of experiment in order to find the motor specification. These calculations will tell us what is the maximum angle, speed, acceleration and weight provided by the base. Due to these calculations, the current will be computed as well.

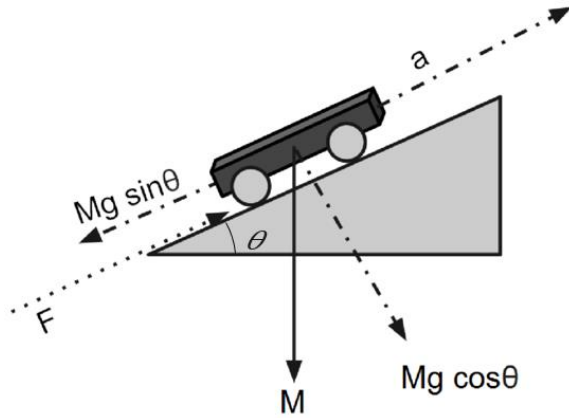


Figure 5.5: Torque calculation.

$r$  = Wheel radius.

$M$  = Platform mass.

$\theta$  = maximum assumed incline angle.

$F$  = Fraction force that produces the torque.

$a$  = acceleration.

$N$  = number of the wheels.

The force result on the base axes:

$$\begin{aligned}
 Mg \cdot \sin\theta + Ma &= F_{friction} \\
 Mg \cdot \sin\theta + Ma &= T/r \\
 T &= \frac{(a+g\sin\theta) \cdot M \cdot r}{N} \quad \text{Equation 5.1}
 \end{aligned}$$

The result should be multiplied by  $100 / \text{The motor efficiency}$

The base characteristics are:

$r = 4"$ ,  $M = 64\text{Kg}$ ,  $a = 0.3 \text{ m/s}^2$ ,  $\theta = 32^\circ$ ,  $N = 4$  and the motor efficiency is 68.5%. Using the torque equation 5.1, the result torque is:

$$T = 13 \text{ Nm}$$

The torque of the RS555 Motor - 12V is 0.2059 Nm and the angular velocity is 7750 rpm. Additionally, the maximum torque that the P60 gearbox can handle is 35 ft-lb (47.4 Nm) [16], which means it is higher than what our design needs. By using P60 Gearbox with 64:1 gear ratio, the result:

$$T = 0.2059 * 64 = 13.177 \text{ Nm.}$$

The wheels velocity is

$w_2 = 7750 / 64 = 122.18 \text{ rpm}$  ( $122.18 / 60 \times 2 \times \text{Pi} = 12.79$ ), where  $w_2$  is the angular velocity of the wheels.

Not that the angular velocity will be transformed to a linear velocity because there are a losses due to wheels rollers slippage on the ground (For more information refer to wheels section and appendix A). Therefore, the effective rotational velocity of the Mecanum wheels is:

$\dot{\theta} = n\dot{\theta}_m \sin\alpha$  [17], where  $\dot{\theta}_m$  is the motor velocity and  $n$  is the gear ratio.  $n\dot{\theta}_m$  is already calculated in the previous step.  $\alpha$  is the orientation angle of rollers on the wheels.

$$= 122.18 \times \frac{1}{\sqrt{2}} = 86.39 \text{ rpm} \quad (86.39 / 60 \times 2 \times \text{Pi} = 9 \text{ rad/sec})$$

$$86.39 \times 2 \times \text{Pi} / 60 \times 4 \times 2.54 / 100 = 0.92 \text{ m/s. (inches converted to meters)}$$

### 5.1.1.2 Current calculation

Power:  $P = T \cdot \omega_2$

The angular velocity  $\omega_2$  in rad/sec

$$P = V \cdot I \quad V = 14.6V$$

Then,  $I = T \cdot \omega_2 / V = 13.177 \times 12.79 / 14.6 = 11.54$  A is the maximum current.

We can conclude from the previous calculation that the gear will handle the required torque. Additionally, the resultant torque which is 13Nm is for the ultimate situation. For instance; if we change the weight to 40kg and the incline to 10 degree, the resultant torque will drop to 2.96 Nm, and the current will drop to 2.59 A. There is a simple tool in the robot shop website to perform all these calculations (<http://www.robotshop.com/dc-motorselection.html>).

### 5.1.1.3 Battery Capacity Calculation

The current drawn by each part of the robot:

- Platform motors = 12 A.
- Waist Actuator = 10 A.
- One Kinects devices = 1 A.
- One Monitor = 1 A.
- One Arm = 3 A.

- One Arduino board = 1 A. The total A = 28A.

Based on these calculation, the chosen battery was a four cell pack battery. Each cell is 3.65V and they compose a total of 14.6V. Additionally, the capacity of each cell is 60 ampere per hour.

### **5.1.2 Robot Design**

The MCECS-Bot robot is shown in Figure 5.6. The robot base is consists of four Omni-wheels which are connected to four DC motors via gear boxes. The robot is constructed from aluminum bars assembled together using brackets, bolts, nuts and washers. The maximum expected weight of the robot is about 143lb (65kg). I created a 3D model using google Sketch Up program. The result of my work can be found in google 3D warehouse:

[http://sketchup.google.com/3dwarehouse/details?mid=2d5e490d368bede3ed1231cfd3b598dd &prevstart=0.](http://sketchup.google.com/3dwarehouse/details?mid=2d5e490d368bede3ed1231cfd3b598dd&prevstart=0)

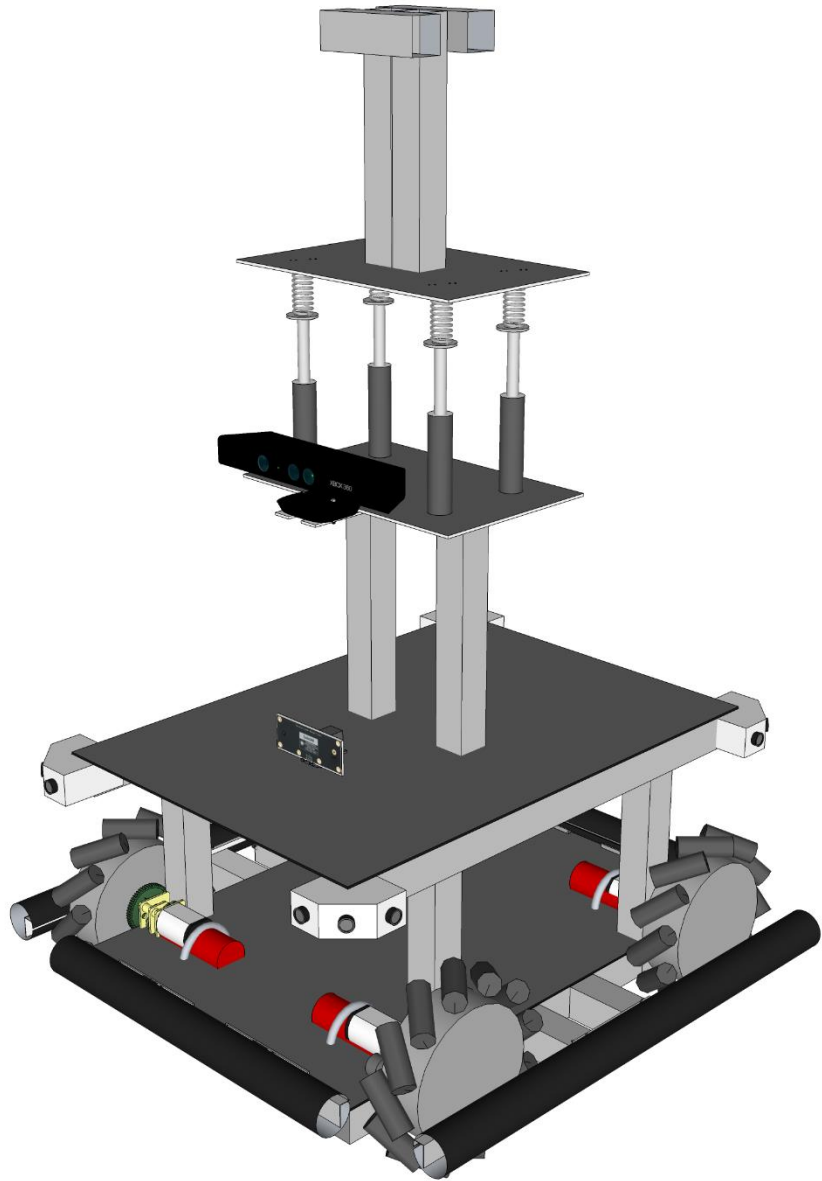


Figure 5.6: PSU Guide Robot [18].



### 5.1.3 Construction Parts

#### 5.1.3.1 Aluminum Bars

There are several shapes of aluminum bars that have been used in this project. There is the square shape (Figure 5.7) with .5"x.5" and thickness of 0.065" which weighs 0.13lb per foot, 1.5"x1.5" and a thickness of 0.065" which weighs 0.45lb per foot and 1.5"x1.5" and a thickness of 0.125" which weighs 0.732lb per foot.



Figure 5.7: Square shape aluminum bar.

There is the L shape (Figure 5.8) with 0.5"x0.5" and a thickness of 0.065" which weighs 0.068lb per foot, .5"X.5" and a thickness of 0.125" which weighs 0.131lb per foot, 0.75"X0.75" and a thickness of 0.125" which weighs 0.205lb per foot and 1.25"X1.25" and a thickness of 0.125" which weighs 0.343lb per foot.

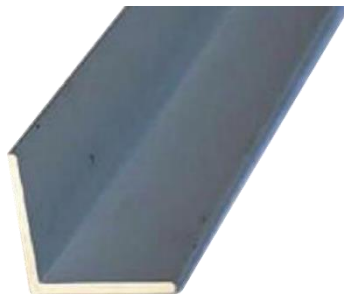


Figure 5.8: L shape aluminum bar.

### 5.1.3.2 Bolts

There are two types of bolts. The first one is a stainless steel socket screws which is 2" in length and 3/16" in width. It is shown in Figure 5.9.



Figure 5.9: Socket screw bolt.

The other one is stainless steel machine screws which is 2" in length and 3/16" in width. It is shown in Figure 5.10.



Figure 5.10: Machine screw bolt.

These two bolts are used to put the aluminum bars of the main structure together.

### 5.1.3.3 U bolt

This U bolt (Figure 5.11) is 4-1/2" in length and 2-1/4" in width. This U shape bolt is used to hold the gearbox to the aluminum bar, so the gearbox and the DC motor will not detached from the base during the robot movement. Figure 5.12 shows how the U bolt is used in to attach the motor and its gear to the main frame.



Figure 5.11: U shape bolt.



Figure 5.12: U shape bolt is used to attach the DC motor and its gear to the main frame.

#### 5.1.3.4 Nuts

The lock nuts (Figure 5.13) are nylon which will tighten the bolts strongly. It is 3/16" in width to support the available bolts.



Figure 5.13: Nylon nut.

### 5.1.3.5 Brackets

There are several types of stainless steel brackets with L shape. There is 1"X1/2" with 2-holes, 1-1/2"X5/8" with 4-holes, 2"X5/8" with 4-holes, 1-1/2"X3/4" with 4-holes, 1-1/2"X1-1/2" with 6-holes and 2 1/2"X1-1/2" with 6-holes. There is 4"X4" T shape with 5-holes. All of these different brackets that were used to build the main structure are shown in Figure 5.14.

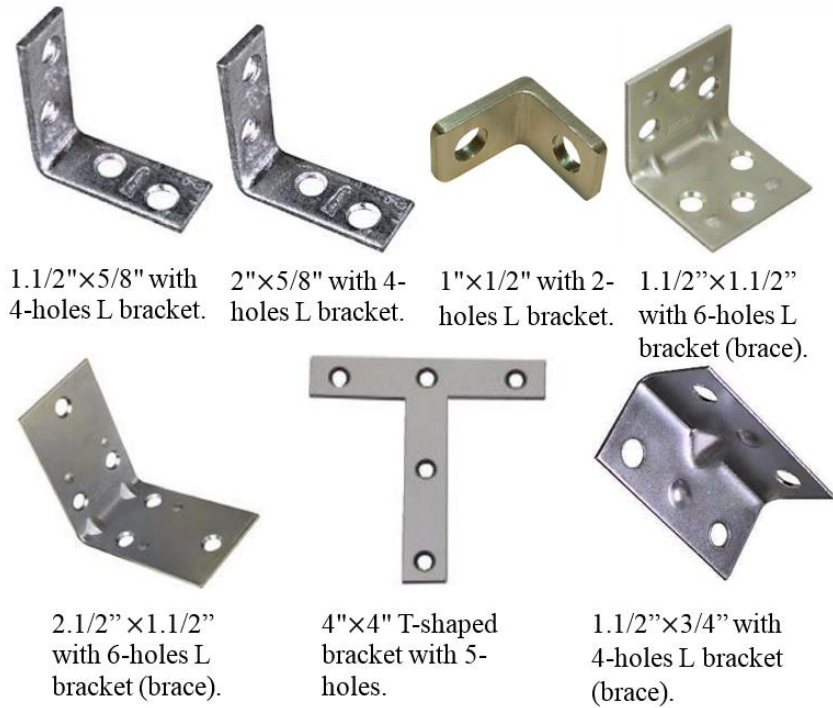


Figure 5.14: Different brackets that were used in the main frame.

### 5.1.3.6 Washers

There are 8 flat washers, that each 2 of them are connected to one gearbox to prevent the hub key and the spacer from falling off the shaft (as shown in Figure 4.16). Washers are shown in Figure 5.15.



Figure 5.15: Flat washer.



Figure 5.16: Motor spacer and its holding washer.

### 5.1.3.7 Spacers

#### 5.1.3.7.1 Bronze Self-Lubricating Spacer

The outer diameter of this bronze spacer (Figure 5.17) is  $\frac{3}{4}$ " and the inner diameter is  $\frac{1}{2}$ ". It is used to space the key hub in the gearbox since the gearbox shaft length is  $1\frac{1}{2}$ " and the key hub is 1" (as shown in Figure 5.17). The available one in the market is 1" in length, so it has to be cut in order to fit in the gearbox.



Figure 5.17:  $\frac{1}{2}$ "  $\times$   $\frac{3}{4}$ "  $\times$   $\frac{1}{2}$ " bronze spacer.

#### 5.1.3.7.2 Aluminum Spacer

These spacers (Figure 5.18) are used to fill in the extra length in bolt that is used to attach the gearbox to the Aluminum bars underneath it. Because the length of the available

bolts is 2” and the height of the aluminum bar is 1.5”, these spacer must be used in the MCECS Bot design.



Figure 5.18: Aluminum spacer.

#### 5.1.4 Base Parts

##### 5.1.4.1 Wheels

There was a discussion among all teams that work on the MCECS-Bot project about the type of wheels that are going to be used in the MCECS-Bot guide robot. The debate was about choosing either Omni or traditional wheels. After discussing the advantages and disadvantages of both, the Omni wheels were chosen due to their ability to derive in any direction instantaneously. Table 5.1 shows the comparison between the normal and Omni wheels [19].

Conventional wheels	Omni wheels
Wheels in aligned the body	Wheels are 45° with the body
2 degrees of freedom	3 degrees of freedom
High load capacity	Good for small and light robots
Simple design	Complex design

Table 5.1: Conventional vs. Omni wheels.

Afterward, Mecanum wheels came in the picture and the debate started again. The Omni wheels (Figure 5.19) are connected diagonally by 45 to the base and can provide movement to any direction. On the other hand, the Mecanum wheels (Figure 5.20) are connected in alignment to the base and can provide the same movement. Table 5.2 shows the comparison between Omni and Mecanum wheels [19].

Omni-Wheels	Mecanum Wheels
mounted diagonally (45°)	mounted parallel to the body
move in any direction (3 DOF)	move in any direction (3 DOF)
rollers are in the side of the of the wheel	rollers are constructed diagonally in the wheel
useful for small and light robot	can handle heavy objects

Table 5.2: Omni vs. Mecanum wheels.

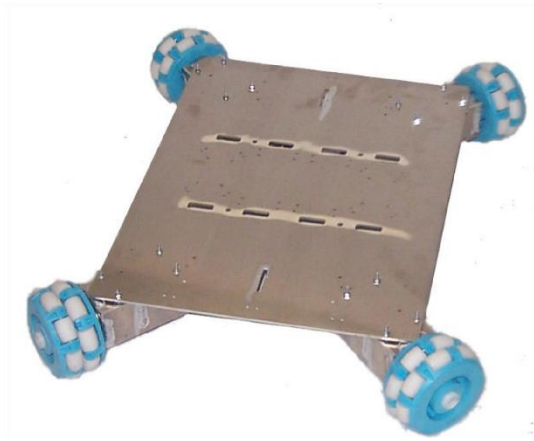


Figure 5.19: Omni Wheels alignment [20].





Figure 5.20: Mecanum wheels alignment [21].

The settlement was on the Mecanum wheels because the good looking and endurance hence they can endure more weight than Omni wheels. Also, unlike the Omni wheels, the Mecanum wheels have the rollers aligning diagonally so it will provide the movement that the Omni wheels provide to the robot. Mecanum wheels are special wheel designs that are based on a concept that achieves traction in one direction and allows greater flexibility in congested environments [22]. The Mecanum wheels are 8” in diameter and 2” in width and each weighs about 2.5lb. See appendix A to learn more about these wheels.

The degree of freedom between Omni and Mecanum wheels does not matter, since they both have three degrees of freedom. Table 5.3 [19] shows the advantages and disadvantages of the Mecanum wheels.

Advantages	Disadvantages
Compact design	Very complex conceptually
High load capacity	Discontinuous wheel contact
Simple to control	High sensitivity to floor irregularities
Less speed and pushing force when moving diagonally	Complex wheel design

Table 5.3: Mecanum wheel advantages and disadvantages.

The Mecanum wheel movement profiles are shown in Figure 5.21.

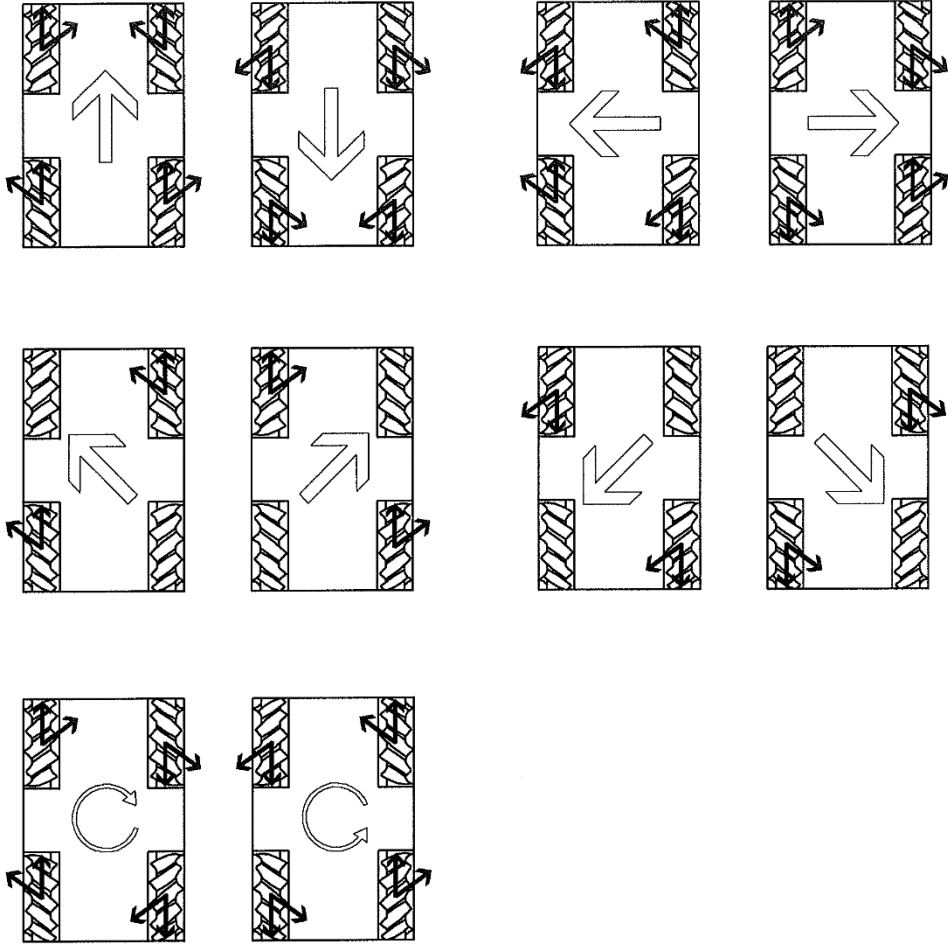


Figure 5.21: Mecanum wheel movement profiles [23].

**5.1.4.2 DC Motor Controller (RoboClaw 2x15A.)**

Basic Micro’s RoboClaw motor controller can control a pair of brushed DC motors using serial, RC, or analog inputs. Integrated dual quadrature decoders make it easy to create a closed-loop speed control system. This version can supply a continuous 15 A per channel (30 A peak) [24]. Figure 5.22 shows a picture of this micro controller. In this robot, two of these motor controllers are used to control four DC motors. There is a set of mini

switches by which different functions can be implemented (More information can be found in Appendix B). Interface setting, the size of the battery pack and other specification can be determined using these mini switches. Moreover, encoders can be connected to these motor controllers in order to implement the odometry system. See Appendix F for more information about the system circuit boards. <http://www.pololu.com/catalog/product/1496> supports all the documentations required to work with this motor controller as well as the libraries of codes related to that purpose.

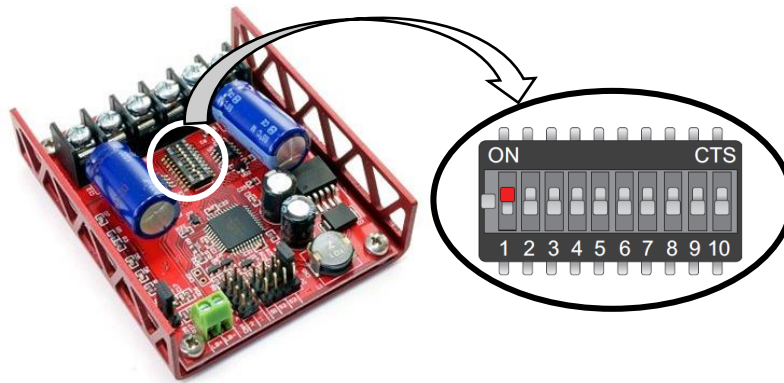


Figure 5.22: RoboClaw 2x15A [24].

### 5.1.4.3 DC Motors

There are 4 of RS555 brushed DC motors (Figure 5.23) that will run the wheels. DC motors are operated by 12V and controlled by two RoboClaw motor controllers. They are connected to the Mecanum wheels via gearboxes. Table 5.4 shows DC motors specification. To see more on DC motors, refer to appendix C.

Specification	Values
Voltage (V)	5-15
Nominal Voltage (V)	12
No load speed (rpm)	7750
No load current (A)	0.4
Stall current (A)	15
Stall torque (mN-m)	205.9
Efficiency (%)	68.5

Table 5.4: DC Motor specification.



Figure 5.23: DC motor [25].

#### 5.1.4.4 Gearbox

The P60 RS-540/550 gearbox (Figure 5.24) has a ratio of 64:1. It is the medium between the DC motor and the Mecanum wheels and will provide the speed and torque required to move the robot. It comes with a pinion that should be mounted to the Dc-motor shaft (Figure 5.25).



Figure 5.24: Motor gearbox and the required pinion to attach the motor to the gear [16].

#### 5.1.4.5 Encoders

This encoder sends a 200 pulse per rotation. Using this encoder allows us to know the direction and the speed which allows adding a closed loop feedback controller. The encoder is shown in Figure 5.25. Appendix B has more information about these encoders.



Figure 5.25: Encoder [26].

Features [26]:

- Resolution: 200 Pulse/Rotation
- Input Voltage: 5 - 12VDC

- Maximum Rotating Speed: 5000rpm
- Allowable Radial Load: 5N
- Allowable Axial Load: 3N
- Cable Length: 50cm
- Shaft Diameter: 4mm

#### **5.1.4.6 Encoder Gears**

A Tetrax gears are used in this design to transfer the movement from the wheels to encoders. Tetrax gears are expensive as they cost \$17-19. I found another source of gears which is more affordable. <http://www.servocity.com/index.html>. This website is a good resource to find motor accessories like gears, hubs and bearing, etc. In order to use these gears, the bore of the 3" diameter gear had to become 0.5", so we made them bigger by using a 0.5" drill bit. After that we used one of the holes that surrounding the bore with the mechanical key. Using the mechanical key is to attach the gear to the gearbox shaft. Figure 5.26 shows the gears and how they are attached to the encoder as well as the gear shaft.



Figure 5.26: gears to attach the encoder to the gear shaft.

#### 5.1.4.7 500 Hub Key

This hub key (Figure 5.27) is used to attach the wheel to the gearbox using small machine key (Figure 5.28), which I found in the robotics lab. Moreover,  $\frac{1}{2}$ " spacer is used to fill in the extra space in the shaft.



Figure 5.27: Gearbox 500 hub key.



Figure 5.28: Gearbox key.

#### **5.1.4.8 Arduino**

The Arduino mega (Figure 5.29) is the intermediate processor between the main brain (PC) and the motor controller. It has the main code to control the Mecanum wheels. It sends the commands to the DC motors via DC motor controller. Additionally, it sends the LRF measurements to the main PC. Another Arduino is used to gather the other sonars data and send them to the main brain. A third Arduino is used to control the waist. Table 5.5 shows some of the Arduino specifications. To learn more about Arduino refer to [www.arduino.cc](http://www.arduino.cc).



Specifications	Values
Operating Voltage (V)	5
Input Voltage limits (V)	6-20
Digital I/O pins	54 (of which 14 provide PWM output)
Analog I/O pins	16
DC current per I/O pin (mA)	40

Table 5.5: Arduino specifications.



Figure 5.29: Arduino Mega.

#### 5.1.4.9 Battery

The battery needed to provide the necessary voltage and current has a nominal voltage of 12V. The battery capacity (Ah) is chosen based on the mass of the robot, the velocity of the robot, the maximum incline, the supplied voltage, the desired acceleration, the desired operating time and other components power consumption. The velocity, the supplied voltage, the desired acceleration, the desired operation time and the total

efficiency will not change. In other words, these variables will be mostly constant and will not exceed their limit. The remaining properties are the total mass and the maximum incline. If we take an extreme situation as 65kg and the incline is  $28^\circ$ , the desired battery capacity will be around 50Ah. The battery in this case will weigh 15kg which may add some weight to the robot and its dimensions are 18.34X16.51X17.5cm which is quite big.

However, the robot may not reach this situation since the robot is made of aluminum which is light and the maximum incline in the engineering building is not that steep. In this case, if we take the mass as 40kg and the incline as  $10^\circ$ , the battery capacity pack will be around 12Ah. The battery weight is about 4kg and its dimension is 15X9.8X9.5cm. This is for the base dc-motors consumption. For the battery capacity of the whole body, refer to capacity calculation section of this thesis.

There is another crucial specification of the battery which is the battery cycle. It means the number of charge\discharge cycles of the battery can experience before it fails to meet specific criteria. In our situation, the desired battery cycle life is 500. A four LiFeMnPO<sub>4</sub>/60AH are used in this robot. There are many reasons of using these batteries. Reading the specifications will show that these batteries have very good features. Figure 5.30 shows the battery pack. More information about LiFePo<sub>4</sub> batteries and their safety user manual can be found in appendix E.



Figure 5.30: LiFeMnPO4/60AH Battery pack [27].

Specifications [28]:

- Nominal Voltage: 12.8V (4X 3.2 V).
- Nominal Capacity: 60 Ah.
- LiFeMnPO4 chemistry.
- Operation Voltage Range: 11.2 to 14.4V.
- Weight: 9.2 kg or 20.3 lbs.
- Dimension: 125X280X180 mm or 4.9X11X7.1 in.
- Max Charging Current: 3C.
- Max Discharge Current: 3C (continuous) / 10C (pulsed).
- Cycle Life : >1500 (80%DOD).
- Operating Temperature: -20 to 65 C or -4 to 149 F.
- Self-Discharge Rate: <3% monthly.

#### 5.1.4.10 Ultrasonic sensors

In our design, 12 ultrasonic sensors (LV-MaxSonar-EZ0 which is shown in Figure 5.31) are used. Figure 5.32 shows how these ultrasonic sensors are attached to the base of the MCECS-Bot. They are numbered from 1 to 12 and these numbers are the same in the wall following code. The main task of these sonars is to detect the surrounding environment of the robot. This will allow the robot to have perception and able to make decisions like avoiding obstacles and navigate itself through obstacles. There are three sonars on each corner of the robot base (on for each direction). The twelve sonars will cover 360 degree around the robot. The sonars detect objects between 0 inch and 254 inch. There is no blind spot, but any object closer than 5 inches is measured as 5 inches. For more information refer to Appendix G. The sonars' specifications are listed below [29]:

- 42kHz Ultrasonic sensor measures distance to objects
- Read from all 3 sensor outputs: Analog Voltage, Serial, Pulse Width
- Virtually no sensor dead zone, objects closer than 6 inches range as 6 inches
- Resolution of 1 inch
- Maximum Range of 254 inches (645 cm)
- Operates from 2.5-5.5V
- Low 2.0mA average current requirement
- 20Hz reading rate
- Small, lightweight module
- Widest beam of the LV-MaxSonar-EZ sensors

- Great for people detection applications



Figure 5.31: LV-MaxSonar-EZ0 [29].

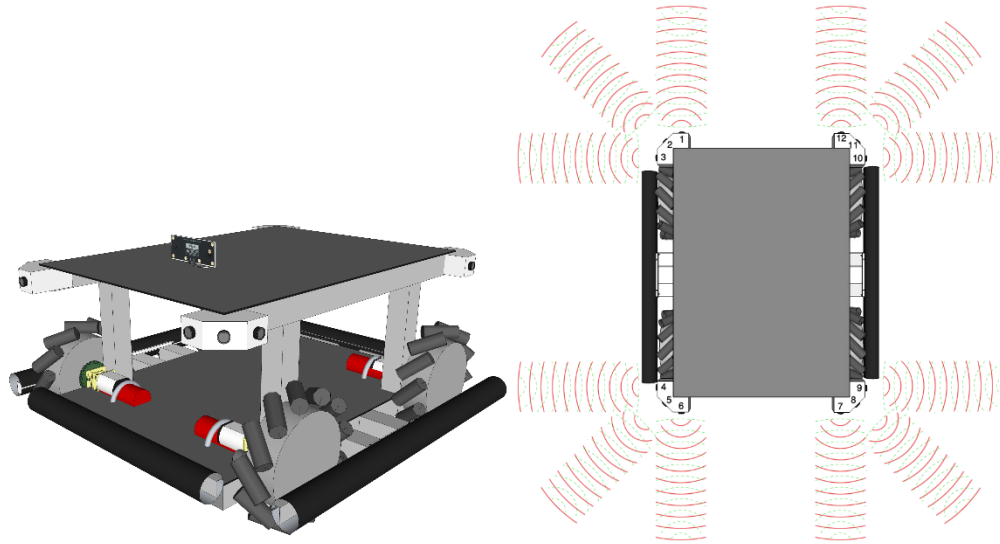


Figure 5.32: Sonars distribution on the robot.

We needed to mount three sonars on each angle as I designed the robot. Mitch Barton created a 3D module to be mounted on each corner of the base. Figure 5.33 shows the 3D module that holds three sonars on each corner of the robot's base.

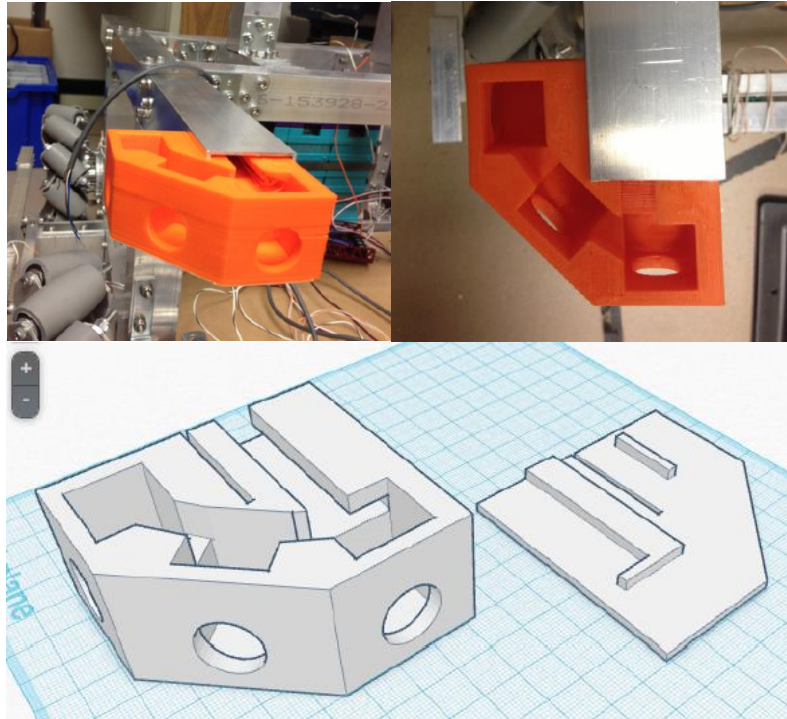


Figure 5.33: Sensors' holder [30]

We got these sonars on Spring 2012 but they weren't attached until Fall 2012. The team that worked on the MCECS-Bot at that moment connected the sonars directly to the power and their data lines to Arduino Uno board. The sonars at that moment weren't reliable and very noisy (See Figure 5.34). I used this measurement in the fourth chapter, in the one dimensional Kalman Filter example. After sending email to the manufacturer regarding this issue, they reply to us with a good explanation to that phenomena. The explanation is that the sonars are always connected to Vcc and GND and this makes them active and send signals all the time, which makes them interfere with each other. There is no signal that would coordinates their work which makes them producing a high aquatic noise affecting each other measurements. In order to test that "theory" we (Mathias Sonardi

and I) put a tape on all sonars and tried to get measurements from only one sonars which left uncovered. The result was much better than when they were uncovered. As soon as we remove the tape from only one sonar, the two sensors started to interfere and make noise. To avoid that, these sonars must work either simultaneously or sequentially.

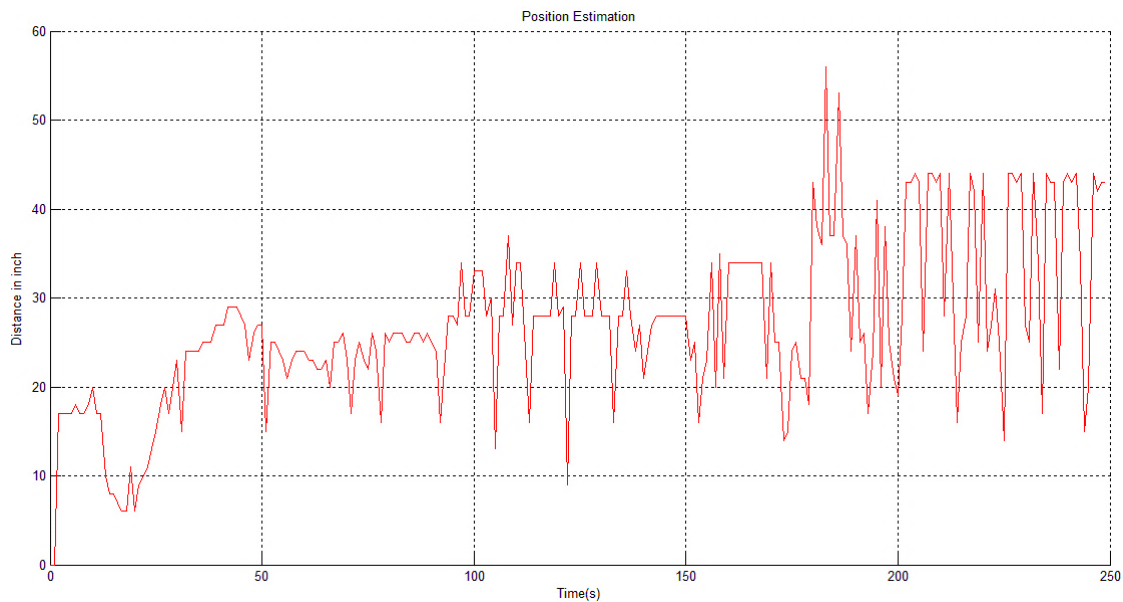


Figure 5.34: Measurement taken from the front sonar. It was taken when the sonars are connected directly to the power without a signal to coordinate their work. It is obvious how noisy the signal is. Kalman Filter is used in the fourth chapter to refine this signal.

In our case, we decided to make the sonars to work sequentially. They must be connected in a method called the chaining method. This method worked when the number of sonars were six but it did not work when we connected all the sonars together. The way that chaining method works is that there is a signal connected to RX pin of the first sonar and it goes from TX of the first sonar to the RX of the second sonar as shown in the Figure 5.35. This signal comes from Arduino Uno board. When the system starts working the signal going to the RX of the first sonar must be held high for 20  $\mu$  second. This will trigger

the chaining method and make all the sonars to work sequentially. I believe that when the number of the sonars increased, the seventh sonar in the chain will get power before getting the coordinating signal. I experimented with the sonars trying to find a solution for this issue because this problem wasn't listed in the sonars documentations. Anyway, I found that splitting the power source for the sonars into two groups (six sonars each) fixed this problem. The first group of sonars was turned on right after the system is on. Then, after 50 m second, the second group of sonars was turned on using the delay circuit. The delay circuit is shown in Figure 5.36. Figure 3.37 shows the measurements of the twelve sonars after the chaining mode was used. The measurements were taken while moving a board back and forth in front of some of these sonars. In appendix F, a diagram of the delay circuit will be found.

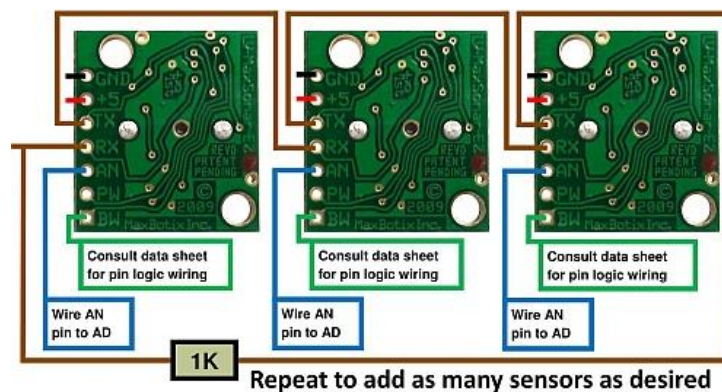


Figure 5.35: Chaining method [31].



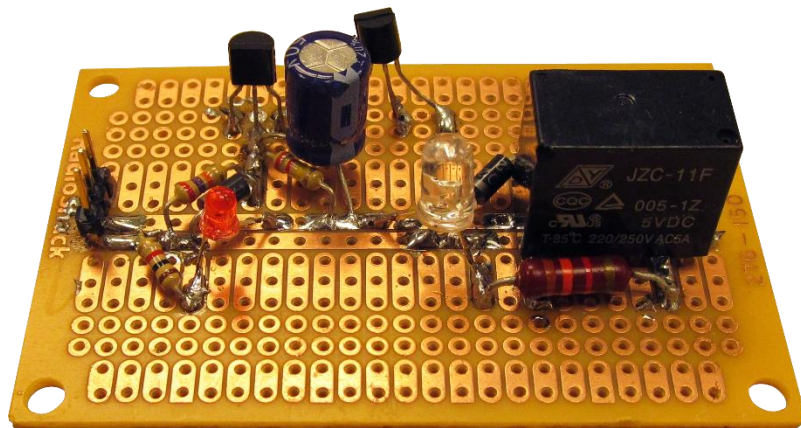


Figure 5.36: Delay circuit. The second group of sonars (six sonars) is connected to it.

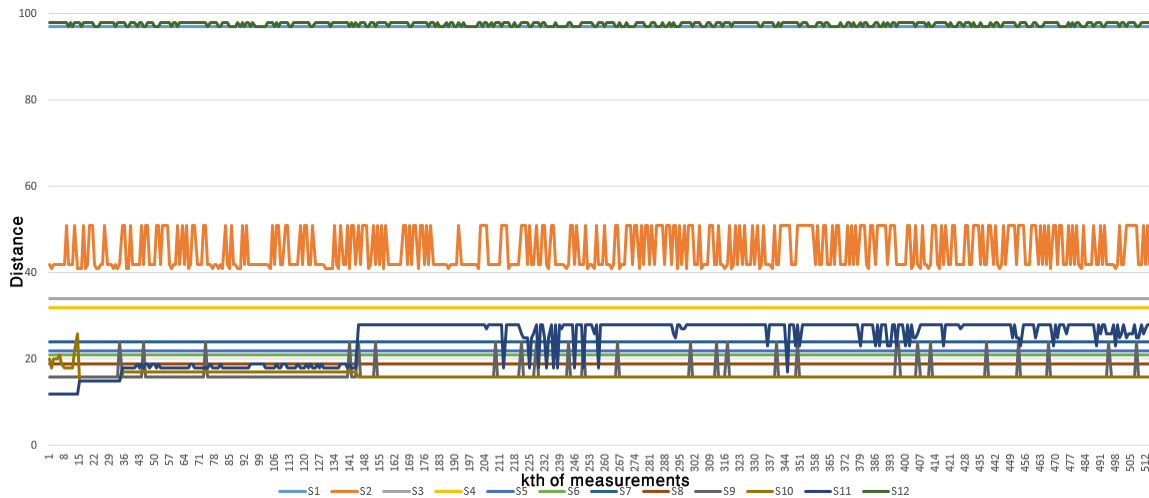


Figure 5.37: The measurements of twelve sonars. It is obvious that the measurements are smooth and there is not noise except some sensors that show some noises. We were testing the sonars by moving a board in different directions and also test the angle that can be covered by each sonar. This test shows that the sonars are reliable and sensitive to any movements or object that comes into their range.

Finally, after all these improvements, we got another issue after the motors started running. When the robot started to move, every time we tried to get the sonars measurements, there was one of the sonars returning the value of zero inch. This is not

possible as the sonars should return 5 inches which is the minimum value. Obviously, the motors and their Motor controller boards are a great source of noise. The signals that comes from the sonars are small and they are affected easily by other noises. One way to solve this problem is to separate the power lines of the motors from the signal lines of the sonars [32]. Another solution is to replace the wires with shielded ones. I recommend to move the wires from the lower level of the base to the second level of the base. Currently, Mathias tried to fix it by changing the code. He prevent the measurements if one of the sonars returns zero. This worked fine and sometimes caused delay because the robot does not receive new measurements to take action.

#### 5.1.4.11 Laser Range Finder (LRF)

LRF is a distance sensor based on laser principles (Figure 5.38). The LRF works on the principle of optical triangulation. It fires the laser on a specific target, then there is a camera that receives the returning laser beam. As shown in Figure 5.39, when the distance from the target changes, the angle on the focal plane changes too. Knowing the change on the camera side allows to calculate the angle ( $\theta$ ). The distance between the camera and the laser diode is already known which is  $h$ . Now, it is easy to calculate the distance  $D$ .



Figure 5.38: Laser Range Finder [33].

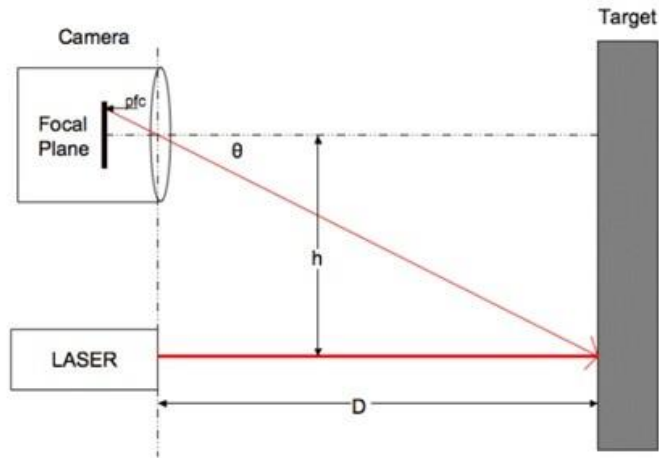


Figure 5.39: Optical triangulation [34].

LRF is attached to a plastic component and mounted on a servo motor in order to cover a range of 220 degree on the front of the MCECS-Bot. Figure 5.40 shows the LRF mounted on a servo motor using the plastic component, which was designed and printed on a 3D printer by Mitch Barton. It is also used in the navigation program with the MRPT library as it simulates the Kinect input.

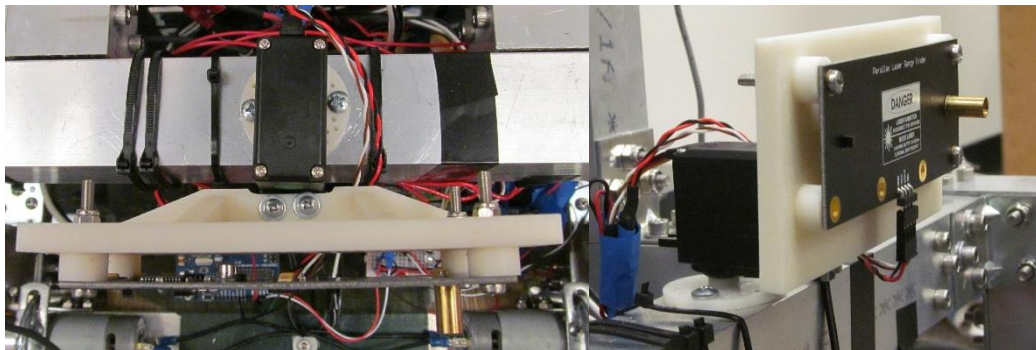
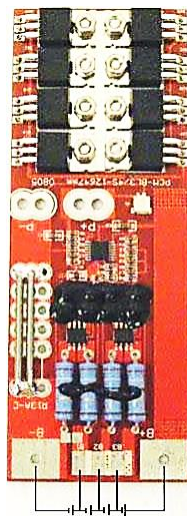


Figure 5.40: LRF mounted on server.

For more information about the circuit diagram, refer to appendix F.

#### 5.1.4.12 Power Management Circuit (PMC)

Using Power Management Circuit (As shown in Figure 5.41) is important because we are using a battery pack of 4 cells. The cells are exposed to different temperatures while they operate. This differences in temperature as well as the differences in the chemical component as they are not perfectly match each other will create voltage differences from cell to cell in time. Using the PMC protects the cells from overcharge/overdischarge, over current drawn as well as keeps the pack cells having the same voltage. In appendix E, the PMC specification and user manual are found [35]. The circuit diagram can be found in Appendix F.



Battery 1 → Battery 4

Figure 5.41: PMC [36].

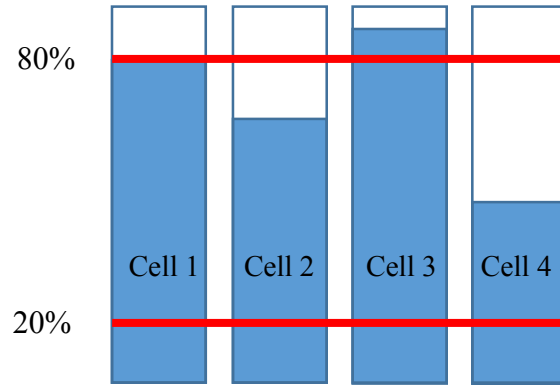


Figure 5.42: Four cells battery pack.

Let's for a moment imagine the four cells in the Figure 5.42. If there is no Power Management Board (Battery Management System) to balance the cells, every time the system will turn off when the fourth cell reached the 20%. The Figure shows that the third cell is fully charged. Therefore, any further charge for the other cells will damage the third and the first cells with over charging and vice versa.

#### 5.1.4.13 Charger

The charger that we used has Over Voltage Protection, Short Circuit Protection and Output Reverse Protection. This charger is a special charger for the lithium ion batteries (charger is shown in Figure 5.43) [37]. It can support 10 Amp which means the charging time is 6 hours. (Charging Time=  $(1.41 * \text{Ah rate of the pack}) / 10\text{A}$  charge current) this formula shows that the time is a bit longer as there are two stages of charging. The first one is when the current is constant and the voltage is increased. On the other hand, after this

stage is the stage when the charger starts holding the voltage and starts decreasing the current till it reaches zero. For more detail see appendix H.



Figure 5.43: Charger [37].

#### 5.1.4.14 Power Distribution Board (PDB)

The power distribution board is one of the most important components in the electrical circuits of the MCECS-Bot. It is important because it distributes the power to all different parts of the robot. The PDB is connecting the battery to the robot and the battery to the charger. When the charger is connected to the PDB, there are five relays that are used to disconnect the robot circuits from the battery. There are four output lines providing the robot's circuits with the 14.6 voltage. The PDB was built by Phil Lamb in Winter 2013. I fixed couple technical issues with PDB. I fix the problem of connecting the charger to the PDB because it did triggered the relays. I also changed some rails to get the fifth relay to work correctly. Additionally, I added a 12V regulator to be used to reduce the voltage from

14.6V to 12V, which is used by Kinect or the Monitor. Finally, I added many LEDs to indicate the status of the PDB and its output. Figure 5.44 shows the Power Distribution Board.

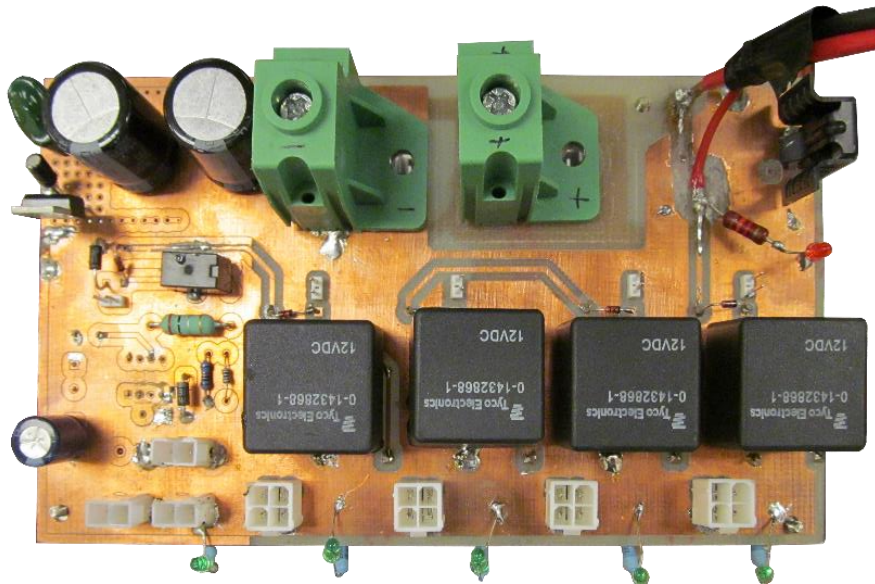


Figure 5.44: Power distribution Board.

For more information about PDB, appendix F contains the PDB schematic diagram.

#### 5.1.4.15 Bumpers

The bumpers are important to our robot as there still exists a dead zone which can't be recognized as an obstacle by the Kinect or ultrasonic sensors. The main goal is to not use these bumpers because we should have an efficient system that is able to detect the obstacles and not to crash on them. The bumpers are used as an emergency stop in case there are no inputs from the other sensors. Two micro-switches (Figure 5.45) are used in

each bumper. Springs as well as spacers are used to protect the micro-switches from being damaged during operation. Each micro-switch is simply operated as they work either normally open or normally closed. For instance, when it hit something it will simply turn off the input from +5 to the analog/digital input in the Arduino board. Figure 5.46 and Figure 5.47 show the bumpers. I added simple code after connecting the bumpers to the right pins in Arduino Mega (The number of pins are given in appendix F). This code is used to detect if the bumpers were triggered or not, and if so it will stop the robot immediately. For instance, if the robot is moving forward and it bumped into some obstacle, it will stop immediately. However, the other directions remain still functional, which means that the robot will be able to move right, left or back if the front bumper is triggered. The code is found in appendix I.



Figure 5.45: Micro switch.



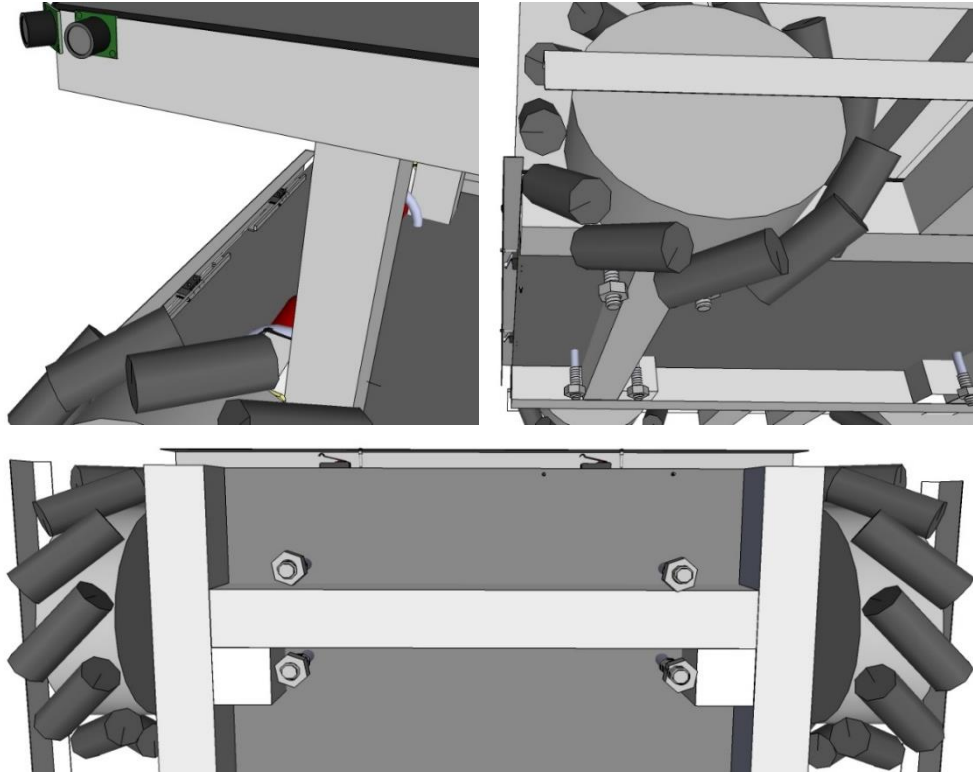


Figure 5.46: Limit switches that are used in the bumpers.

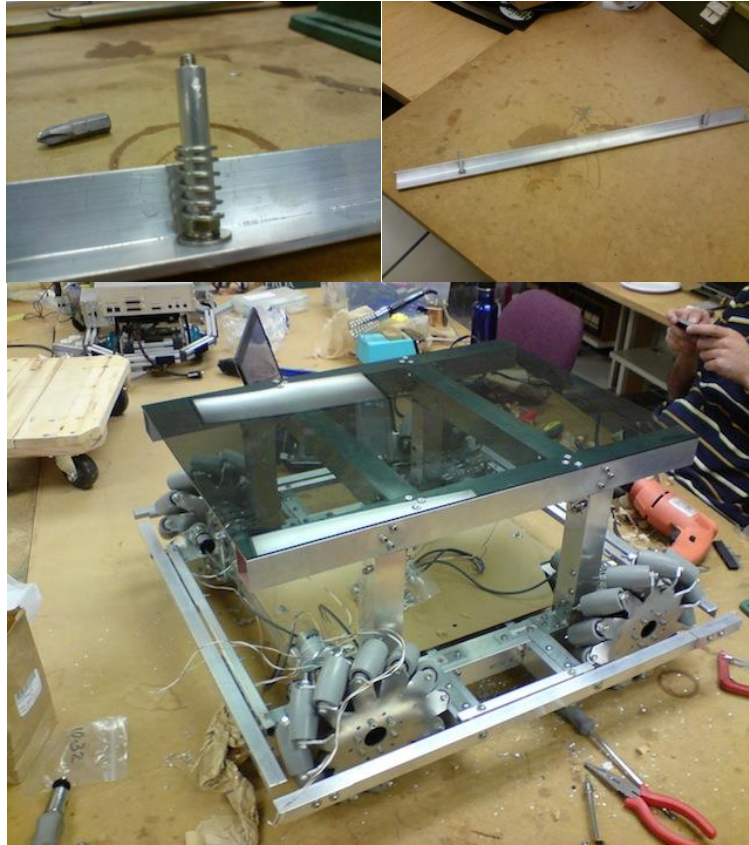


Figure 5.47: These images show the steps of constructing the bumpers.

### 5.1.5 Actual Work

After having all parts available to build the base (taking into account that it took more than 2 days to find the right hardware materials) we started by calculating the aluminum bars dimensions. The base dimension should be 17"X 24" in addition to the width of each Mecanum wheel which is 2" as shown in Figure 5.48.

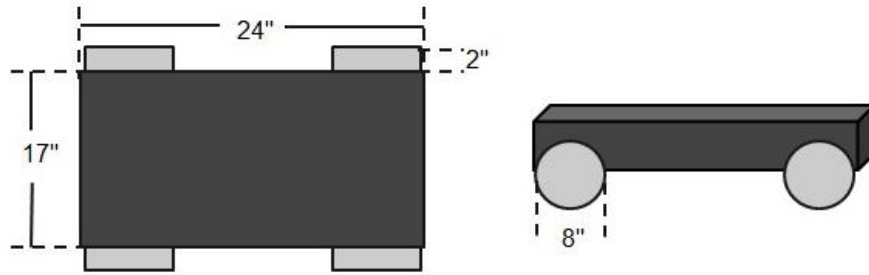


Figure 5.48: Base dimension.

Then we started to cut the bars using the right blade that we bought for the lab miter saw (Figure 4.49 shows the construction of the Aluminum parts). After that, we drilled the bars to make holes by using a drilling press that is available in the shop of the robotics lab and we spent more than 2 days to finish cutting and drilling them. By using the brackets and bolts, we assembled the aluminum bars together (Figure 5.50).

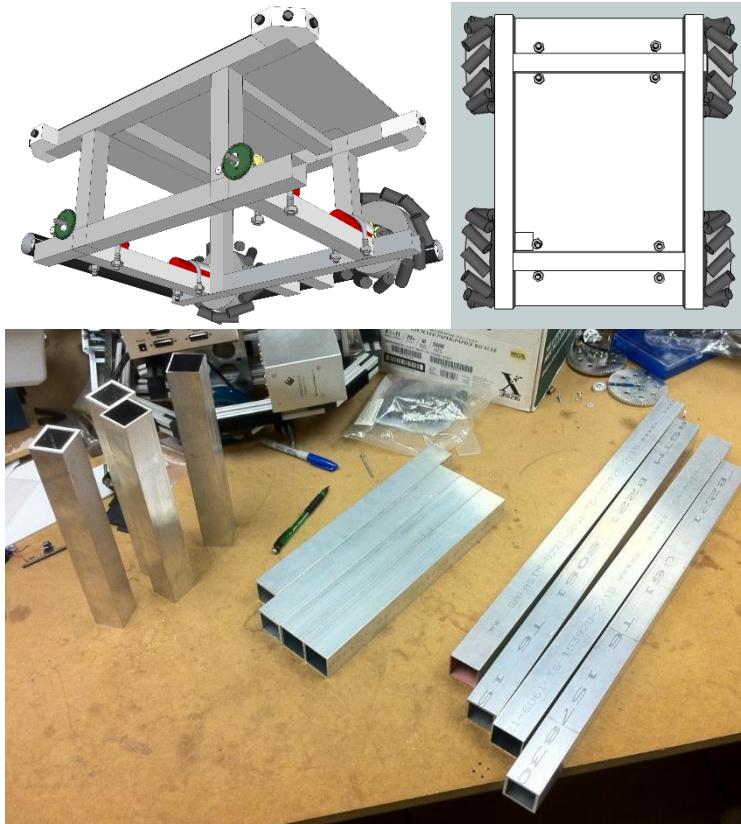


Figure 5.49: The construction of Aluminum bars.

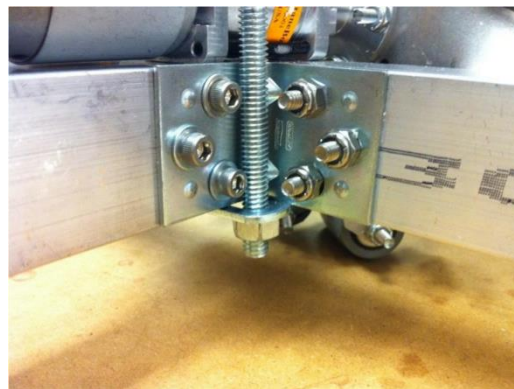


Figure 5.50: Brackets holding two bars.

Before assembling the wheels together with the DC motors and gearboxes, the gearbox is needed to be greased, and the wheel is needed to be assembled (took around one day). After doing that, we connected the gearbox with the DC motor, then to the wheel using the 500 key hub. The same procedures were done for remaining wheels, gearboxes and DC motors (took one day).

The next step is to mount the assembled group of DC motor, gearbox and wheel on the aluminum bars structure (Figure 5.51). This was done by making another two holes in the aluminum bars for the gearbox, and using U shape bolts to hold the combination to the bars (Figure 5.52).



Figure 5.51: Mecanum wheel is connected to the gearbox and DC motor and mounted on the aluminum bar.



Figure 5.52: U shape bolt holds the gearbox to the main frame.

## 5.2 Waist

The waist was built by David Gaskin in the Spring 2012. I contributed to the waist as I helped in choosing the motor controller and the structural construction. The waist consists of four linear actuators (as shown in Figure 5.53). The linear actuators will give the robot's waist simple movements of human's waist like bending forward/backward and left/right as well as bending in diagonal direction. The waist is located on top of the legs and carries the neck and arms.



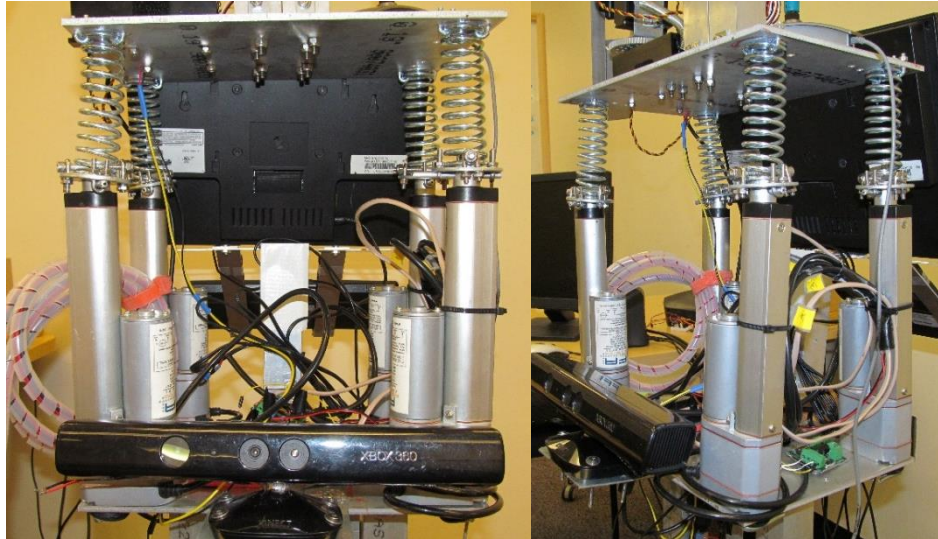


Figure 5.53: MCECS Bot's waist.

### 5.3 Arms

Ultimately, there should be two arms setting on the waist. Each arm has 6 degrees of freedom. Currently, there is only one arm. The structure of this arm is made of aluminum and six servo motors (one in each joint). Minimaestro board is used to control this arm.

Figure 5.54 shows the arm.

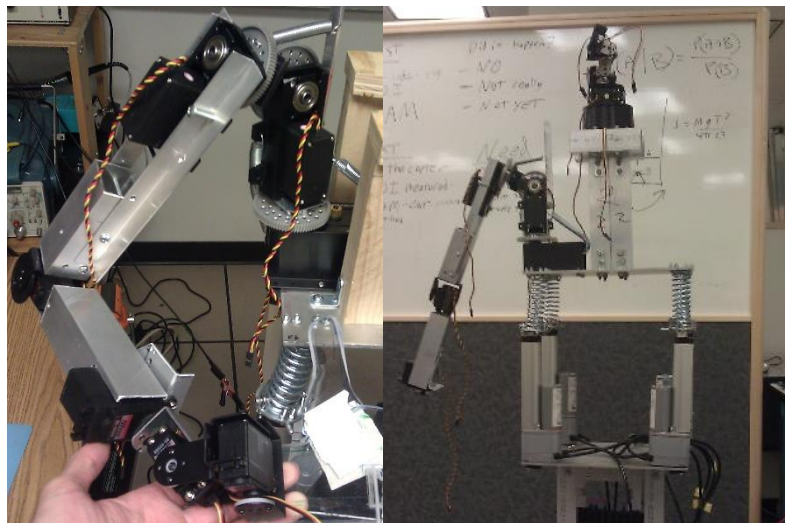


Figure 5.54: Robot's Arm.

## 5.4 Neck

The neck has two degree of freedom representing the agreement and disagreement gesture of the human. The neck is shown in Figure 5.55. There are two stepper motors in it. It is controlled by the same Minimaestro board that is used to control the arm.

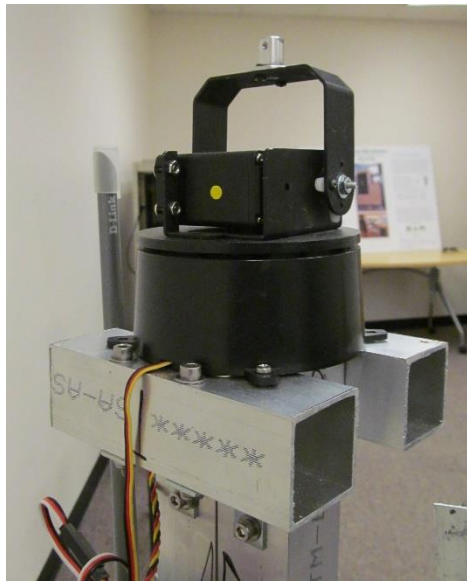


Figure 5.55: MCECS-Bot's neck.

## 5.5 Head

The robot is intended to look like a butler. A Samsung Galaxy tablet is used as the butler's face. Simply, the tablet will show the gestures of the eyes and the mouth like blinking and smiling. Students from the art department are involved to build the aluminum mask that will cover the tablet. Figure 5.56 shows the head of the MCECS-Bot.



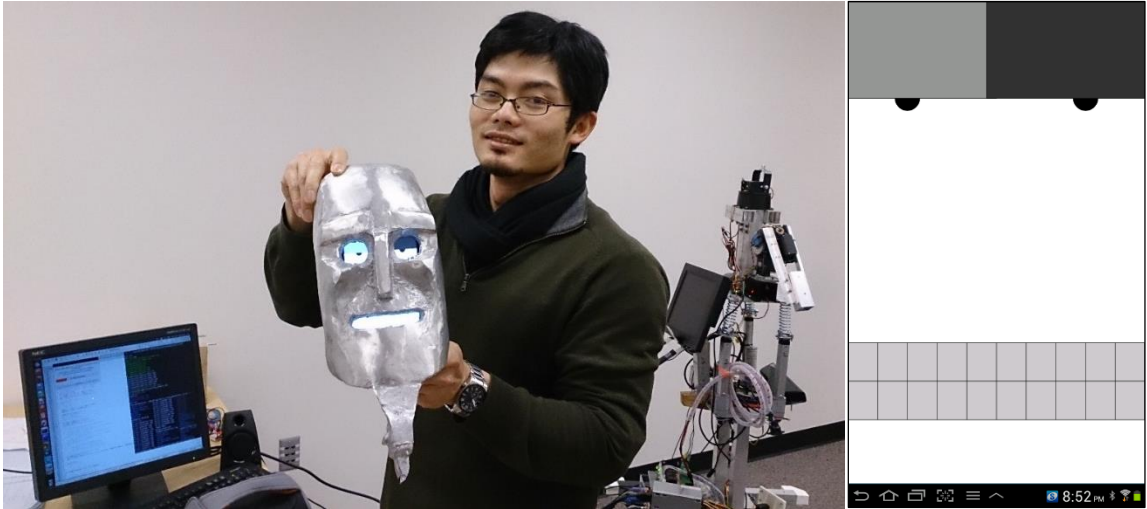


Figure 5.56: The head consists of an aluminum mask that covers the tablet which represents the head<sup>8</sup>. On the right, a screen shot from the tablet shows the face of the butler.

---

<sup>8</sup> In this picture is Mathias Sunardi who is the team manager and he designed the head.

## 5.6 Software System<sup>9</sup>

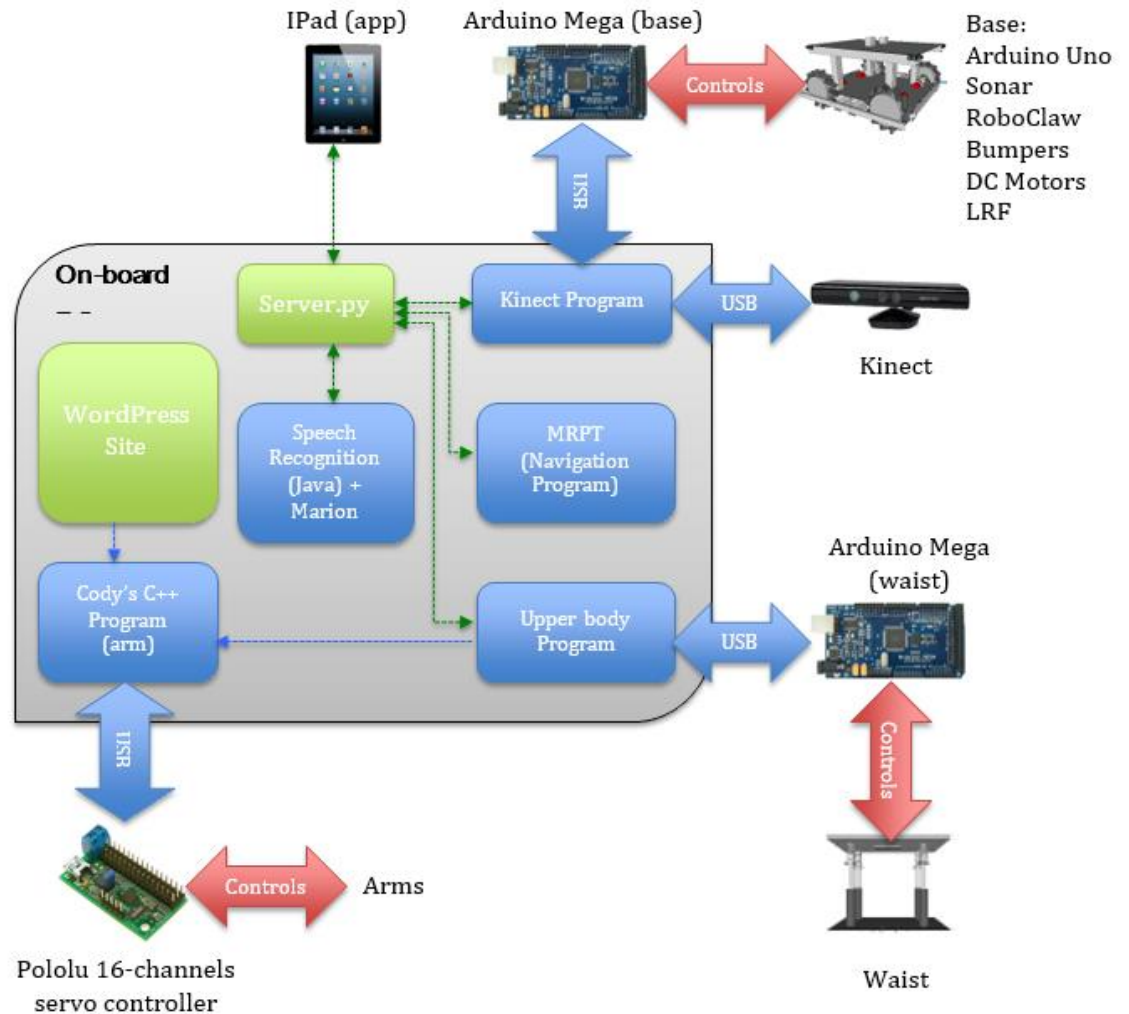


Figure 5.57: MCECS-Bot's software diagram.

<sup>9</sup> This section is part of Mathias Sunardi's report "MCECS-Bot Progress Report Summer13". Except the MRPT subsection which was part of my thesis.

### 5.6.1 Overview

We shall give a brief overview of the system here. Details of the system will be explained in the subsequent sections below. Figure 5.57 shows the different software components that currently control the MCECS-Bot and how they communicate/connected with each other. There is a Server program written in Python (Server.py) that MUST be run FIRST when activating the robot. The Kinect, MRPT, Upper body, and Speech Recognition (Java)+Marion programs connect to the Server program as clients (dashed green arrows). Once connected, any of the client programs can send messages/commands to the other client programs through the Server program. The Server program will broadcast any message it receives from any of the client program to ALL the clients connected to it. It is the duty of each client program to decide whether to act upon the message/command. To facilitate this decision-making process, we devised a simple messaging format/protocol which allows easy parsing to determine which client sent the message, which client the message is for, and what command is being sent (explained below).

The Server-client program uses the TCP protocol. The main reason we use this method is because the TCP protocol is one of the established standards for network communication hence is platform-agnostic, and the Server program can run as its own process as to not interfere with the operation of each software component. Furthermore, the different software components are written in many different programming languages, and it is very difficult to consolidate using native code (e.g. using wrappers); the Kinect and upper body programs are written in Processing language, the iPad app is written in Objective-C, Cody Hank's gesture control program was written in C++, the

MRPT/Navigation program is written in C++, the Speech Recognition+Marion program was written in Java, and the Server program itself is written in Python. Luckily, all these programming languages are equipped with libraries that include the necessary methods for TCP communications.

The software components mentioned above are the ones that have close control over the robot's interaction and movement functions: moving the base, gesturing, tracking, etc. The WordPress site is a website (set up by Cody Hanks) that serves two functions. The first function is to provide a user-friendly interface to update the faculty, labs, and other information database. Currently, we have not employed a DBMS (Database Management System) such as PostgreSQL, MySQL, or SQLite3. Our 'database' is a set of WordPress *pages*. We'll talk about this component in more detail below, but for now it is sufficient to state that in the future, this database should be made appropriate by using a DBMS.

Except for the iPad app/program, all the software components run on the on-board PC of the MCECS-Bot. The on-board PC is running under Ubuntu 12.10 64-bit operating system.

The report is organized as follows: We will start with the Server program and the messaging protocol, the Kinect program, the iPad program, and the WordPress site. We do not describe in detail the MRPT/Navigation program, the Speech Recognition+Marion program, and Cody Hank's gesture control program since there has not been any significant changes in them. Moreover, we have not yet integrated the MRPT/Navigation program into the current system. Although we integrated the Speech Recognition+Marion program into the current system, we are currently not using it since we are using another Sphinx-based

speech recognition library in the iPad app (which will be explained below). For Cody Hank's gesture control program, we were able to implement it without any changes to the program itself except for changing the serial port name.

### **5.6.2 The Server Program: Server.py**

The Server program was adapted from the online tutorial to create a chat client app in iOS/Objective-C [<http://www.raywenderlich.com/3932/how-to-create-a-socket-based-iphone-app-and-server>]. It uses the Python Twisted library, which provides Python classes and methods to program network communications for web applications such as TCP, UDP, SSH, IRC, and FTP. The current Server program uses TCP to receive and send messages to its clients.

Twisted was built upon a design pattern called *Reactor Loop*. As seen in the figure below, the Reactor Loop essentially is a loop that waits for events, if an event is detected, handle the event by performing some tasks. Once the task is done, wait again for the next events (Figure 5.58).

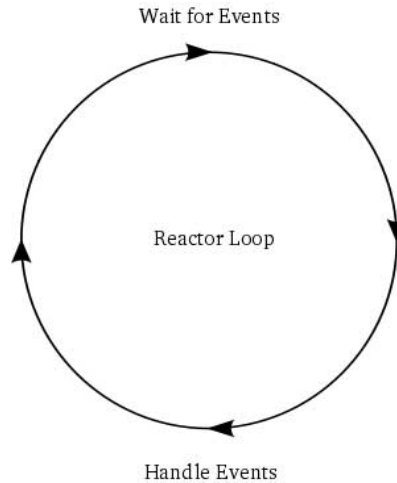


Figure 5.58: Reactor Loop.

There are three objects used in the server program: Factory, Protocol and reactor. When a client connects to the server, a Protocol object is created for the client. The Factory object is responsible for creating the Protocol objects and adds the client to the list of clients (i.e. Protocol objects) in a list variable called “clients” in the Factory object. We create an object called IphoneChat as a child object of Protocol. When a client connects, a new IphoneChat object is created and added to the clients list in of the Factory object. The reactor object will run the loop above and “listen” for any events from any of the clients, whether it is the first time the client connects to the server, when the client is sending a message to the server, etc. How each event is handled is defined in the IphoneChat object definition.

### 5.6.3 Kinect

The Kinect program is written in the Processing language, which is like a subset of Java. The program was first developed by Rami Alshafi in Fall term 2012. His full report can be seen here:

[https://projects.cecs.pdx.edu/projects/roboticsclub-mcecsbot/wiki/Interactive\\_Kinect](https://projects.cecs.pdx.edu/projects/roboticsclub-mcecsbot/wiki/Interactive_Kinect)

Since then, several features have been added to the program, and we have done some refactoring to make the source code easier to read. The Kinect program does the following:

- Provides the “vision” for the robot
- Provides a user interface for the user:
  - o A menu (i.e. a set of buttons will appear on the screen when the Kinect program detected a hand)
  - o Using the buttons, user can interact with the robot using hand gestures: playing music, controlling the movement of the base
  - o Gives feedback to the user on the current state/mode of the robot
- Directly interface and controls the operation of the Arduino Mega for the base via USB
  - o Send commands to the Arduino Mega for moving and stopping the base
- Stops the base if an object or obstacle is too close using the depth image.

Some of the additions done over Summer 2013 are the following:

- Collect information of the state of the base including the distance information detected from the twelve sonars, bumpers, and Laser Range Finder (LRF).
- Send all the information from the Arduino Mega on the base and from the Kinect program itself to the iPad.

#### **5.6.4 MRPT Code (Navigation Program):**

MRPT is a Mobile Robot Programming Toolkit. It comprises a set of C++ libraries and a number of ready-to-use applications [38]. Figure 5.59 shows the MRPT libraries. The user should provide MRPT with a map of the environment that his experiment is running on and whatever information used to update the system status. This information can be come from different sets of sensors that should be supported by MRPT. It supports a range of sensors like camera, sonars, LRF, laser scanner, infrared and gyroscope.



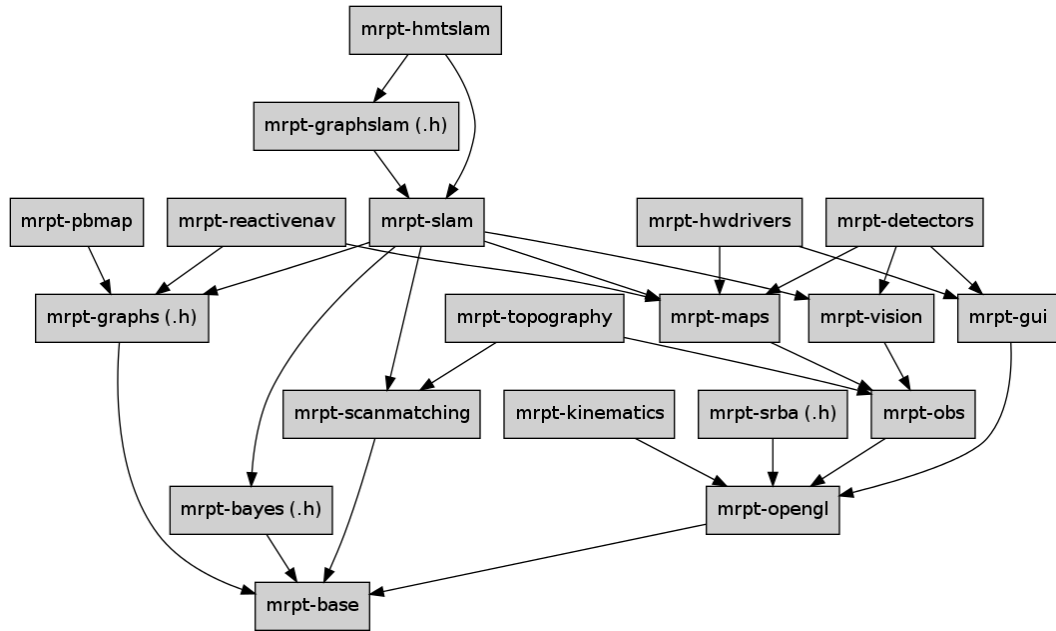


Figure 5.59: MRPT Libraries [38].

There are different applications and classes provided and supported by MRPT in order to be implemented by the users based on their needs (Figure 5.58). One of these classes is Extended Kalman Filter, which included in the following class:

`mrpt::bayes::CKalmanFilterCapable`<sup>10</sup>

This class contains the system state vector and the system covariance matrix, as well as a generic method to execute one complete iteration of the selected algorithm [39]. In our case, I had to derive all the equations related to the state transition model and to the

---

<sup>10</sup> More information can be found on Kalman Filter page of the MRPT website; [http://www.mrpt.org/Kalman\\_Filters](http://www.mrpt.org/Kalman_Filters)

observation (measurements) model. Additionally, the Extended Kalman Filter class requires all the matrices that related to the system equation.

## **5.7 Overall Circuit Connections**

This section is a summary that illustrates the connections between the power circuits and other controller boards. Moreover, this section is important because it shows the readers how everything is connected and helps them to visualize the robot circuits. In the current chapter, many boards and circuits were presented. Many motor controllers and several Arduino boards are used in this robot. This diversity may confuse readers. Therefore, a general descriptive diagram to the circuit board of the robot is required. Figure 5.60 shows a diagram that illustrates the circuit boards of the MCECS-Bot.

As shown in the figure below, there are twelve sonars connected to the Arduino Uno board which works as a slave to Arduino Mega. Arduino Mega board is connected to the PC to get commands. The main code is set in Arduino Mega1 which has all different kinds of interpretations to different kind of control signals or to send data from sonars to the main PC (MCECS-Bot brain). Any other details were listed in each section accordingly. A detailed schematic diagram for each circuit, Arduino boards connections, motor controllers and power circuits are presented in Appendix F.

MCECS-BOT Block Diagram

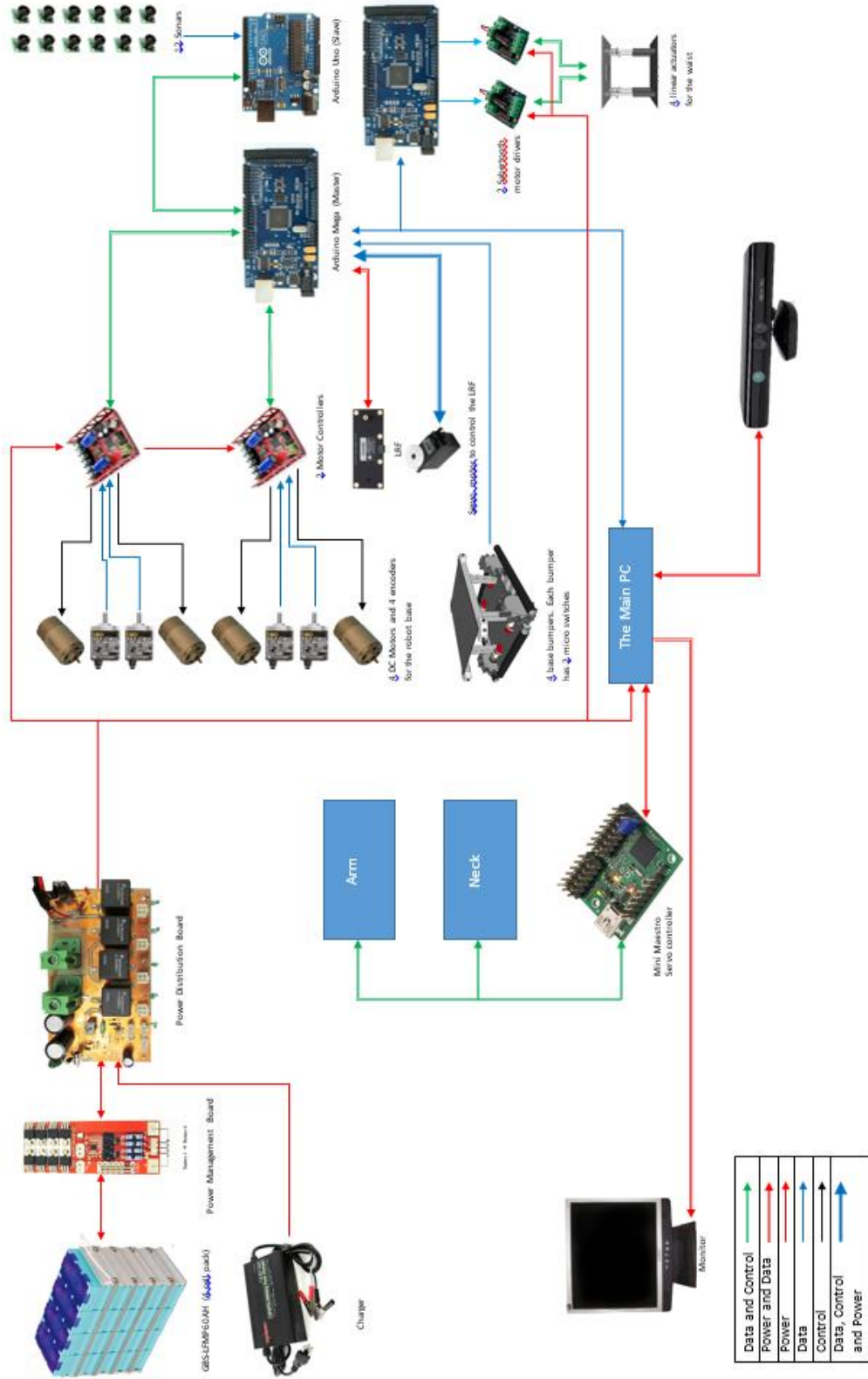


Figure 5.60: MCECS-Bot circuit diagram.

## 5.8 Prices

We looked for the parts that we needed in different places and websites. The prices are varied depending on the quality and the item's characteristics. For example; the dc-motor drivers prices will become 50\$ more if we choose the 26A current instead of 12 A maximum current supplied by it.

### 5.8.1 Base

#### 5.8.1.1 Wheels

There are different wheels varied in the prices depending on the weight that they support or the quality as well. The table below lists different wheels with their prices.

Wheel	Diameter	Supported weight	Price
Plastic Omni-wheel (am-0013)	8"	50 Ibs	34
Aluminum Omni Wheel (am-0098)	8"	50 Ibs	54
Mecanum Wheel (am-0081)	8"	180 Ibs	78
Mecanum Wheel (am-0201)	10"	440 Ibs	180

Table 5.6: Different Wheels characteristics with their prices.

Table 5.6 shows a comparison between the Omni wheels and the Mecanum wheels in prices and characteristics. It is easy to see the marginal difference in prices on one hand and the supported weights on the other hand.

### 5.8.1.2 Aluminum bars

We bought the Aluminum bars from metal market which is near the airport. It is a store that works only on metal stuffs. The materials we needed has different characteristics like the thickness, dimension and shape like square, L shape or cylinder shape. The total cost so far for the aluminum bars is 160\$.

### 5.8.1.3 Gearbox

After looking in different websites, we found that there are a variety of gearboxes. They are shown in Table 5.7.

Gear	Torque	RPM	Price
12V, 58RPM 60:1 Gear Motor	254.8mN.m	58	\$22.58
12V, 17RPM 200:1 Gear Motor	784mN.m	17	\$22.58
12V, 185RPM 20:1 Gear Motor	78mNm	185	\$22.58
Banebots 64:1 P60 Gearbox and RS555 Brushed Motor	13.17 Nm (Combined with the motor)	121	\$64.50

Table 5.7: Different gearbox with their prices.

The Banebots gearboxes prices start at 50\$ for the 4:1 ratio, and end with 80\$ for the 256:1 ratio. Obviously, other motors and gearboxes are not as suitable as using Banebot gearbox and the RS555 DC motor, because their supported torque does not satisfy the MCECS-Bot's weight and the desired velocity.

### 5.8.1.4 Dc motor (RS555 Brushed Motor)

The prices of the dc motor are included with the gearboxes prices. It is about 6\$ each.

### 5.8.1.5 Motor driver

Here are some motor drivers' prices and specifications.

Motor driver	Current	# of motor	Price
Sabertooth driver	10	2	80
Pololu TReX Dual Motor Controller	13	2	100
Bi-directional DC Motor Speed Controller	5-48Amp (modification are needed)	1	25

Table 5.8: Different motor drivers' prices with specifications.

### 5.8.1.6 Arduino

Arduino	Price
Mega	\$59.00
Uno	\$30.00

Table 5.9: Arduino boards prices that used in the base.

### 5.8.1.7 Hardware parts and Robot Components

Item		Specification	Price
Bolts	Hardware stuff 1	Socket and Machine screws	183 for all the hardware parts
Nylon nuts			
Brackets			
Grease			10.08
Washers			1.74
Spacers		Pushing for the gears shaft	6.4 for all 4
Saw blade		For the miter saw	48
Hardware stuffs 2		-----	201.26
		Quantity	
Plastic shield		2	34.32
DC-motor controller		2	179.9
Encoder		4	169.95
Micro switch		10	20.92
BMS board		1	38.41
Gears		2	16.43
Batter pack of 4 cells		1	410.26
Aluminum Bars		----	91.47
Charger		1	66.18
Ultra sonic sonars		12	315.6
PCM (power circuit model)		1	41.36
Total			1835.28

Table 5.10: Hardware prices.

The total cost for this base is (there is half of the Aluminum bars have not been used yet) \$1835.28.

## 5.8.2 Waist

Here are two tables of the purchased materials that David Gaskin listed in his report “Actuators and You, an in Depth Guide” in Spring 2012.

Item	Source	Price	QTY	Shipping	Total
Arduino Mega 2560 Microcontroller rev 3	<a href="http://www.trossenrobotics.com/p/arduino-mega-2560.aspx">http://www.trossenrobotics.com/p/arduino-mega-2560.aspx</a>	64.95	1	0	64.95
Arduino Ethernet Shield	<a href="http://www.sparkfun.com/products/9026">http://www.sparkfun.com/products/9026</a> <a href="http://www.trossenrobotics.com/p/arduino-ethernet-shield.aspx">http://www.trossenrobotics.com/p/arduino-ethernet-shield.aspx</a>	45.95	1	0	45.95
Arduino Box	<a href="http://www.trossenrobotics.com/p/arduino-project-box.aspx">http://www.trossenrobotics.com/p/arduino-project-box.aspx</a>	11.95	1	0	11.95
Pig tail for power	<a href="http://www.trossenrobotics.com/store/p/6612-Barrel-Jack-Female-Pigtail-Lead-2-1-5-5mm.aspx">http://www.trossenrobotics.com/store/p/6612-Barrel-Jack-Female-Pigtail-Lead-2-1-5-5mm.aspx</a>	2.95	2	0	5.9
6vDC power for board	<a href="http://www.trossenrobotics.com/p/power-supply-6vdc-2a.aspx">http://www.trossenrobotics.com/p/power-supply-6vdc-2a.aspx</a>	9.95	1	0	9.95
12v 5A pwr supply for actuators	<a href="http://www.trossenrobotics.com/p/power-supply-12vdc-5a.aspx">http://www.trossenrobotics.com/p/power-supply-12vdc-5a.aspx</a>	19.95	1	0	19.95
Servo wire	<a href="http://www.trossenrobotics.com/store/p/6421-50-Foot-Standard-Servo-Wire.aspx">http://www.trossenrobotics.com/store/p/6421-50-Foot-Standard-Servo-Wire.aspx</a>	12.95	1	0	12.95
Motor Controller	<a href="http://www.trossenrobotics.com/store/p/5104-Sabertooth-dual-5A-motor-driver-for-R-C.aspx">http://www.trossenrobotics.com/store/p/5104-Sabertooth-dual-5A-motor-driver-for-R-C.aspx</a>	59.99	2	0	119.98
Firgelli Actuators	<a href="http://www.firgelliauto.com/product_info.php?cPath=109&amp;products_id=159">http://www.firgelliauto.com/product_info.php?cPath=109&amp;products_id=159</a>	129.99	4	0	519.96
Actuator Mounting	<a href="http://www.firgelliauto.com/product_info.php?products_id=54">http://www.firgelliauto.com/product_info.php?products_id=54</a>	9	4	0	36
Total					847.54

Table 5.11: Parts ordered online.



Item	Cost	Comment
Metal Knockout bit	44.97	For drilling out large metal holes
Dremmel carbide blades	8.98	For shaping holes
Dremmel EZ lock kit	16.53	For locking blades to dremmel
UPS shipping	10.52	For exchanging Motor driver (they sent the wrong part)
Terminal kit with tool	9.04	To make the terminal block
Terminals for 10-12, 14-16 gauge wires	5.58	To make the connections for the terminal block
Battery Clamps	3.29	For connecting terminal to power source
Washers	1.50	For spacing out the actuators on the plate
2x terminal blocks	15.98	For making a easy to connect to power source
14 gauge black wire	1.50	For terminal block and connections
14 gauge red wire	1.50	For terminal block and connections
10 gauge wire	2.35	For battery connections
Safety pins	3.12	For harness connections for aluminum plate and actuators
C-804 compression springs	1.50x4	For joint between waits and top
1/8" cable ties	2.50x20	For securing all the hardware together
3/4" neoprene Grommet	1.21x 4	Replaced the washers, allowing a snugger fit for the actuator mounting
Total	185.70	

Table 5.12: Hardware parts purchased from local hardware stores.

Chapter 6: Literature Review

## **Chapter 6: Literature Review**

In chapter six, whatever information related in literature to and relevant to our project, MCECS-Bot, is presented. It summarizes papers and work of researchers that related to Mecanum wheels. We present information based on literature about Omni wheels or Mecanum wheels, which are a new concept, different from traditional wheels. Secondly, this chapter will present some papers that relate to localization based on Kalman Filters. This will give us a clear idea about what researchers have already achieved in localization approaches. Finally, our robot will be compared to other fellows' robots. There are many guide robots or museum robots. These robots have their pros and cons when compared to our robot. Comparing and appreciating what others have done will show us how far we are still from our intended goal of building a comprehensive and general robotic guide. Material presented in this chapter will give us a detailed presentation of researches outlined in the literature on the subject and how it may be related to our project.

Three main sections will be presented here:

### **6.1 Mecanum Wheels**

Mecanum wheels were invented by Bengt Erland Ilon in 1975. The title of his patent is "Wheels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base". Mecanum wheels have become an interesting research and design subject to many researchers because they have unique characteristics differing them from the traditional wheels. Robots that have Mecanum wheels have advantages in narrow and crowded environments. Mecanum wheels give them more

degrees of freedom to move in any desired direction. More about Mecanum wheels can be found in “section 5.1.4.1 Wheels” in chapter five.

There are not much published about Mecanum and Omni wheels, especially when the research topic is about localization based on Mecanum wheels. The reasons for this is may be because Mecanum wheels are more complicated to control than the conventional wheels. Another reason can be that Mecanum wheels are recently invented and only very recently available in kit form, so that researchers are not familiar yet with their characteristics. However, there are many papers and research reports about controlling Omni wheels and about Omni wheels technical details. Florentina Adăscăliței and Ioan Doroftei [19] presented practical applications of robots that have Mecanum wheels. The characteristics of Mecanum wheels are well presented in their paper. However, they claimed that the slippage of Mecanum wheels prevents the usage of encoders. I believe that it is possible to use encoders, but the designer of four wheels base should be careful if he intended to use encoders. The reason why is because using only two encoders in either adjacent wheels disambiguates the direction of the robot. It will not be possible to differentiate between rotating left/right and strafing left/right. Additionally, putting two encoders in opposite wheels won't solve the problem because it is not possible to differentiate between moving forward/backward and strafing left/right. However, relying completely on encoders is not feasible even with conventional wheels due to an accumulated error. Therefore, using additional sensors are proposed in MCECS-Bot to make the sensors compensate the errors of each other. Publication [19] presented the difference between the Mecanum wheels and conventional wheels. Furthermore, it lists

different improvements on Mecanum wheels to improve the efficiency of wheels as well as overcoming some other disadvantages.

The kinematics and dynamics of Mecanum wheels are presented in Nkgatho Tlale and Mark de Villiers paper [17]. These authors derived the laws to find displacement, acceleration and velocities for different kinds of movements realized by Mecanum wheels. More information about Mecanum wheels are found in chapter five “section 5.1.4.1 Wheels”. In other words, their paper presented only three movement profiles and introduced their kinematics and dynamics. However, more complex profiles can be achieved. The rotational movements (three and two wheels’ rotations) described in the paper would be more specific if they would describe these motions as arc movements, the concept introduced in the previous paper. Additionally, paper [14] addressed all these rotations and analyzed them assuming that all the 4 wheels are rotating with the same velocity. Further complex analyzes are needed for addressing the profiles where the wheels rotate with different velocities producing more combinations of movements (rotations). Finally, the researchers used encoders with other sensors like gyroscope and accelerometer to do their tests. Using encoder with other sensors have been explained in other papers [40]. This goes against the authors’ claim in [19] that using Mecanum wheels prevents using encoders. Encoders can be used but the accumulated error would be more than the one that comes from using encoders with traditional wheels. More sensors should be used to successfully complete the task of controlling a platform with Mecanum wheels. More information can be found in chapter five in section “wheels” as well as in Appendix A.

## 6.2 Localization and Sensor Fusion

The literature is rich with localization and Sensor fusion research topics because localization is one of the most important aspects that robotic researchers are interested in. Localization and mapping as well as SLAM are essential topics in any mobile robot design. Localization is the process of what a robot uses all related sensors to locate itself in a previously known map. On the other hand, mapping is the process of constructing a map using sensors that sense the surrounding environment. SLAM is a combination of the two and it is obviously known from its name, as it stands for Simultaneous Localization and Mapping. The robot constructs the map and locates itself on it. In our case, as stated previously, the map is already known, so we don't have the "lost robot problem" to be solved here. The robot has to localize itself on the map as it performs some required tasks. The problem with all the previous processes is that the sensors are not perfect and there are always uncertainty and accumulated error that comes from different sensors that are used to get the desired measurements. Therefore, filters like Kalman Filters and Particle Filters are used to satisfy the needs of these algorithms. In our project, we explored different kinds of algorithms to solve simple tasks. For instance, a problem of finding the best solution for a robot in a maze. Genetic algorithm and Search algorithm were used to solve such kinds of problems. Later, we decided to explore a deeper algorithms as our guide robot project became more advance. Bayesian estimations was our intended general approach to create algorithms to be applied in our project. First, we started, as stated in the introduction chapter, with the Particle Filter which is part of MRPT code. It was applied on PEOPLE-Bot which was the first prototype of the intended PSU guide robot.

After having MCECS-Bot built, our concern about improving and implementing a reliable algorithm increased. Kalman Filter was the best candidate to be implemented in our robot. It was hard to find articles about localization based on Kalman Filter for Mecanum wheels based robot platforms. Nevertheless, in literature, many researchers have worked on localization using different kinds of Bayesian estimation. The following literature is about localization and sensor fusion for conventional wheels robots. Sergios I. Roumeliotis and George A. Bakey [41] listed a combination of Kalman filter and Bayesian estimation in order to find landmarks in the surrounding environments and to localize the robot position. There are external sensors that provide the robot with the information that leads it to the landmarks. Odometry is used to move from landmark to landmark. To find the best estimation while moving between landmarks, Kalman Filter is used. Bayesian hypothesis testing is used to combine the Kalman estimation and the landmark-detecting algorithm. Based on the information that comes from each landmark, a set of multiple hypotheses is created. This is called a multiple hypothesis testing.

MCECS-Bot has more sensors and its sensors are more advanced and better than the sensors used in the robot presented in this paper. The authors of [31] didn't describe in details what kind of external sensors they used in their experiments. Additionally, they used only odometry as INS (Inertial Navigation Sensors) while MCECS-Bot has them all: odometry, Sonars, LRF and Kinect. I believe that it is good to include this paper as a source in my thesis. However, the technique used on this paper is not directly applicable by us as our system is more advanced.

Songmin Jia et al [42] presented using Kalman Filter estimate based on information coming from LRF and odometry to achieve robot's self-localization (Basically, they fused odometry with LRF). The result of Kalman Filter was compared with the result of using only odometry. Definitely, the result of Kalman Filter was the closest to the real values. Finally, if there is an error in drawing the map, a GUI program using an additional camera will correct it. This system is close to ours, except that MCECS-Bot is a Mecanum wheels platform, which makes control more complicated. In their system the LRF is used to detect the environment while MCECS-Bot has LRF, sonars, Kinect and other inertial sensors from the tablet (Gyro, Accelerometer and Compass). The measurements coming from encoder and LRF were fused in EKF used in their system. This is similar to the method proposed in my thesis, but the LRF is replaced with sonars in MCECS-Bot. Similar approach is followed by Fantain Kong et al [43] in finding the corners and doing localization. Jungmin Kim et al [44] presented a quite similar method. They used sensor fusion on Kalman Filter to find the best estimation of the robot position. In the previous papers and as described in the fourth chapter, sensor fusion is a solution for the sensors' shortcomings. It has been done using Extended Kalman Filter. Simply, information from gyro, Accelerometer and/or encoders are used to figure out the state equation of the system. Then, the measurements model is figured out using the information coming from LRF or sonars. Researchers in [45] and [46] presented few different sensor measurements fusion algorithms and a comparison among them which gives us a clearer understanding about the sensor fusion concept.

Miguel Pinto et al [47] presented a unique method of teaching by using EKF with NXT Lego to get the robot localization. The authors showed how using an interesting kit



like Lego would attract undergraduate students to start their journey with robots. They presented in a simplified way how to use Extended Kalman Filter to find the best estimation of the robot position. It was not a complicated problem as well as this paper is just for teaching purposes. I believe that this is somewhat similar to one of the MCECS-Bot ultimate goals. MCECS-Bot is meant to be a great platform for other students to experiment with and improve in the future.

### **6.3 Other Guide and Museum Robots**

This section presents different museum and guide robots and compare them with MCECS-Bot. The robots chosen for this purpose are one of the most famous robots in the world. In the presented literature in this section, I will not list the software program of the other robots. I will concentrate only on topics directly related to localization. Interaction, speech recognition, facial recognition and gesture recognition are not topics of this thesis. However, MCECS-Bot has all different kinds of interaction. Teams have been working on them and developing them since the project of PEOPLE-Bot started. There exists already a developed software for each specific interaction. There is a tablet to interact manually with the robot in order to give a direct order or to get more information about PSU or different facilities in the Engineering Building as well as faculties. The ultimate goal is to have a remote access to the MCECS-Bot from anywhere around the world using internet connection. This will allow people to get the same information as they would obtain in a one-on-one interaction, which are the same information that I mentioned before. People will have a direct access to the wiki page of the project at [ece.pdx.edu](http://ece.pdx.edu) website. This project

will be an open source which means that students can learn, improve and participate in this project. I believe that this project will be a flagship project at Portland State University and will help Maseeh School of Engineering and Computer Science to recruit new students.

### **6.3.1 RHINO [48]**

RHINO is the oldest robot that I reviewed in the literature. It has a set of five sensors that used for human interaction and for navigation purposes. These sensors are laser, sonars, infrared, camera and tactile. For localization, Markov localization are used to maintain a probabilistic belief as to where the robot currently is located. Markov localization failed to satisfy a dynamic environment because Markov assumption works only in a static environment. RHINO faced troubles in the presence of people which was most of times during its operation in a museum. This is due to the algorithms that were used to maintain its position. The robot doesn't look humanoid. It lacked some important sensors like the encoder. For interaction there was a small keyboard of four buttons. People can push them to choose different options like tour or listen to a previously recorded voice.

### **6.3.2 Minerva [49]**

Minerva is a tour-guide robot which was operated successfully for two weeks in a Smithsonian Museum. During its operation the robot was able to interact with people and move at speed of 1.63 m/sec which is considered a high speed for a guide robot. The designers were able to address different kind of problems like navigation in a dynamic

or/and unmodified environment, short term human robot interaction and Virtual tele-presence which was based on a web interface allowing people to access the robot remotely from around the world, control its movements, and have a streaming live images. In this project the authors didn't address the problem of SLAM. Basically, the authors manually control the robot in the new environment and use three different sensors to build the map. These sensors are laser scanner, camera and odometry. After the map was constructed, the robot will solve the problem of localization. Markov localization with the same three sensors was used to address this problem.

### **6.3.3 RoboX [50]**

RoboX is a tour guide robot. It can navigate itself in a previously known map. The robot has a lower part which is the base in a hexagonal shape. The base contains the main circuits, batteries, two laser range sensors, bumpers and wheels. The upper part consists of the body of the robot like waist and face. The face has two eyes; one of them is a camera, and moveable eyebrows. RoboX has different sensors such as laser range sensors, odometry, camera for face tracking and camera facing the ceiling for localization purposes. Localization uses Extended Kalman Filter to find the best estimation. Bumpers are used for emergency stop.

#### **6.3.4 Maggie [51]**

Maggie is a humanoid robot that has artistic design of a young girl. The base is a ready base from iRobot, which is not constructed by the designers. It has twelve sonars, twelve infrareds and twelve bumpers. The robot is also equipped with a laser range finder. Additionally, there is a color camera for people tracking. The robot can show some expressions by using colored lights as a visual expression. Tactile sensors were added in different places on the upper part of Maggie. The authors claim that the robot has a high quality speech recognition.

### 6.3.5 Comparison between the Presented Robots and MCECS-Bot

	<b>RHINO</b>	<b>MINERVA</b>	<b>RoboX</b>	<b>Maggie</b>	<b>MCECS-Bot</b>
<b>Humanoid</b>	No	Yes	Yes	Yes	Yes
<b>Arms</b>	No	No	No	Yes, 1 DOF <sup>11</sup>	Yes, 7 DOF
<b>Face</b>	No	Yes	No	No	Yes
<b>Manual Interaction</b>	No	No	Yes, using four buttons	No	Yes, using tablet
<b>Localization</b>	Markov localization	Markov localization	Extended Kalman Filter	Not mentioned	Extended Kalman Filter
<b>Wheels</b>	Differential Steer	Differential Steer	Differential Steer	Differential Steer	Mecanum Drive
<b>Sensors</b>	<ul style="list-style-type: none"> <li>•Laser Range Finder</li> <li>•25 Sonars</li> <li>•Infrared</li> <li>•Camera</li> <li>•Tactile</li> </ul>	<ul style="list-style-type: none"> <li>•Laser scanner</li> <li>•25 Sonars</li> <li>•Camera</li> <li>•Odometry</li> </ul>	<ul style="list-style-type: none"> <li>•Laser Range Finder</li> <li>•Odometry</li> <li>•2 Cameras</li> <li>•8 Bumpers</li> </ul>	<ul style="list-style-type: none"> <li>•12 Sonars</li> <li>•12 Infrareds</li> <li>•Odometry</li> <li>•12 Bumpers</li> <li>•Camera</li> <li>•1 Laser Scanner</li> <li>•6 Tactile</li> </ul>	<ul style="list-style-type: none"> <li>•8 Bumpers</li> <li>•Odometry based on 4 encoders</li> <li>•12 Sonars</li> <li>•1 Laser Range Finder</li> <li>•1 Kinect (which has several sensors inside)</li> <li>•Gyroscope, Accelerometer and compass from tablet</li> </ul>

Table 6.1: A comparison between MCECS-Bot and other guide robots.

---

<sup>11</sup> DOF: Degree of Freedom.

Chapter 7: Using Kalman Filter with MCECS-BOT

## **Chapter 7: Using Kalman Filter with MCECS-BOT**

The previous chapters were intended to be an introduction to the current chapter. This chapter is the main chapter of my thesis. The first chapter presented an introduction to the whole work in this thesis. The second through fourth chapters were about Kalman filter. Kalman filter was presented from the basics. Many examples were used to illustrate Kalman filtering in order to easily deliver the ideas to the readers. The fifth chapter was also important because it describes the system that Kalman Filter is applied for. Knowing the field that Kalman filter is applied for is important because the MCECS-Bot is an important project for PSU and this project is important to next PSU robotics' teams for further researches and improvements. Especially important will be testing the first version of the robot with real public that visits our School.

In this chapter, two main algorithms are illustrated. The first one is a simple wall following algorithm based on measurements coming from the sonars. The first part contains two sections. The first one presents a simple algorithm to find the closest wall and follow it. The second section presents an approach in which the measurements from all sonars are refined using Kalman Filter before sending the measurements to the wall following algorithm. Results of both sections will be discussed and compared.

The second part of this chapter is about using the MRPT code based on the Extended Kalman Filter to find the best estimation of the robot's position. Simply, Arduino Mega board sends the measurements coming from sonars (which simulate the Kinect data) and the odometry data to the main PC which contains the main process of Extended Kalman Filter. The original MRPT code is based on Particle Filter and gets the measurements from LRF

and odometry. Sonars will be used instead of the LRF in simulating Kinect measurements because they are faster and more descriptive (the front four sonars will be used in this experiment). Extended Kalman Filter will be used instead of Particle Filter which represent the second part of this chapter. The second part will contain two part that deals with solving the problem of localization; the first part is using one dimensional Kalman Filter and the second part is using two dimensional Kalman Filter.

## **7.1 Wall Following Algorithm**

### **7.1.1 Normal Wall following**

In this task, the robot has to follow the wall. There are more than one sensor to be used in this algorithm. We can use either LRF or sonars. Sonars are faster than LRF because in LRF it takes time to rotate the LRF device to the other side. In my experiment, among the twelve sonars that are mounted on the robot base, only four sonars are used to execute the task of the wall following. Figure 7.1 shows the distribution of the sonars that are used in this task.



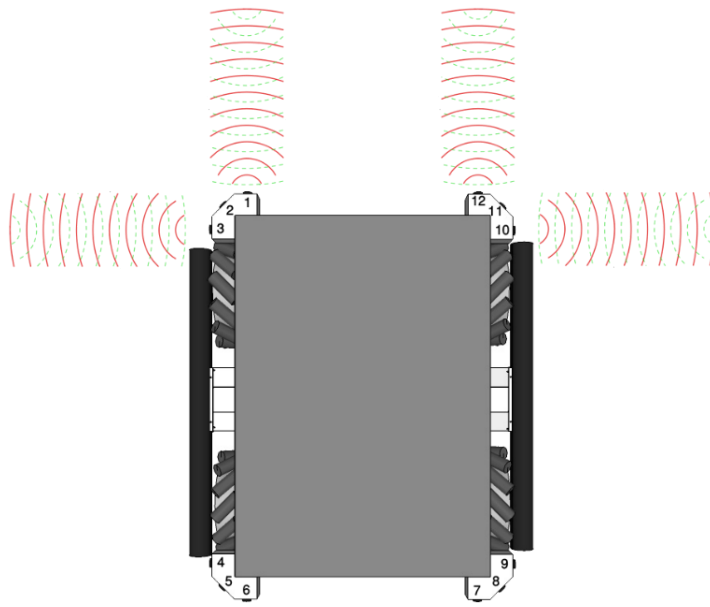


Figure 7.1: Four sonars used in the wall following algorithm.

The algorithm is illustrated in the following flow chart:

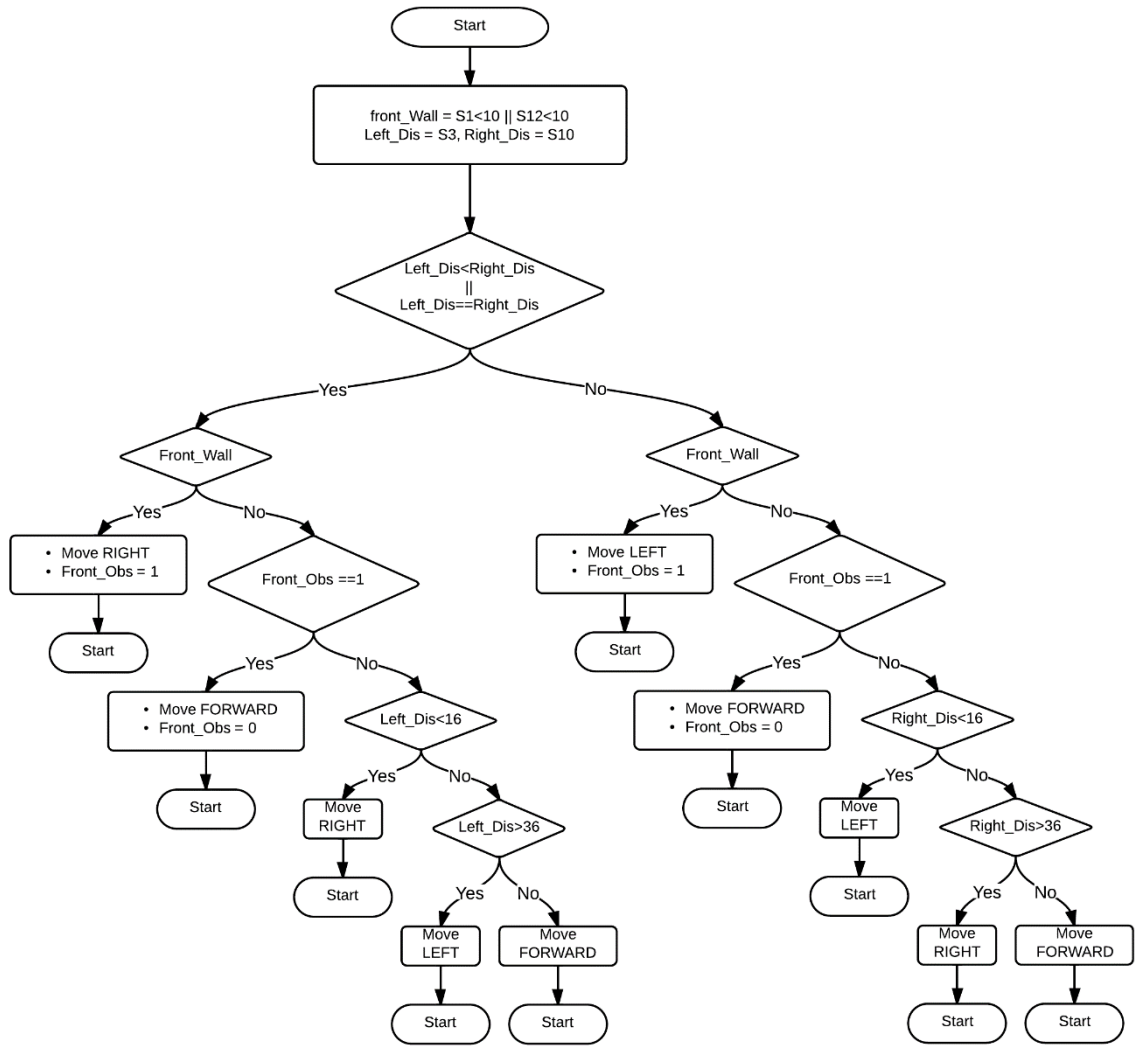


Figure 7.2: Wall following flow chart.

This flow chart was transferred to software code which is uploaded to the Arduino Mega\_1 processor. The code was tested and it worked perfectly<sup>12</sup>. The code of the wall following algorithm can be reviewed in Appendix I.

### **7.1.2 Wall Following with Kalman Filter**

As presented in section “5.1.4.10 Ultrasonic Sonars”, there is still a problem of “zero measurements” coming from at least one sonar when data are acquired from twelve sonars. I stated in that section that Mathias fixed it by adjusting the code. The code will repeat the reading if one of the sonars is giving zero. However, this solution causes delay because the robot kept waiting to get a real data from all sonars. This delay might last for long time which cannot be tolerated in case that we need a fast response from the robot in the wall following. Another solution will require hardware changes like changing the wires or moving them away from the noise source. This solution will be applied in future.

In this section, Kalman filter is applied to refine the measurements that are coming from the four sonars used in the wall following algorithm. Instead of using separate Kalman Filters which means one for each measurement, one variant of our approach uses a Kalman Filter that has a state equation with four state variables. Kalman Filter will look like a four

---

<sup>12</sup> A video available on YouTube shows the wall following test:  
<http://www.youtube.com/watch?v=1FVPdDgY684>

dimensional one, but the variables are independent inside it. These Kalman Filter equations are presented below:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}_k$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}_k$$

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} 25 & 0 & 0 & 0 \\ 0 & 25 & 0 & 0 \\ 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

Figure 7.3 shows the code of Kalman Filter algorithm. It receives the real measurements from four sonars and returns the estimate of each one of them.

```

KalmanFourState
function [dis1, dis3, dis10, dis12] =
Simple1DKalman(z1,z2,z3,z4)
persistent A H Q R
persistent x P
persistent firstRun

if isempty(firstRun)
    A = [1 0 0 0;
         0 1 0 0;
         0 0 1 0;
         0 0 0 1];
    H = 1*eye(4);

    Q = 1*eye(4);
    R = 25*eye(4);
    %The R has a large value so the Kalman will not affected
    the changing measurements.

    x = [20 20 20 20]';
    P = 100*eye(4);
    %The initial value of the error is big to cover the
    unexpected unreal zero measurement which due to the
    acoustic noise.

    firstRun = 1;
end

z=[z1 z2 z3 z4]';
xp = A*x;
Pp = A*P*A' + Q;

K = Pp*H'/(H*Pp*H' + R);

x = xp + K*(z - H*xp);
P = Pp - K*H*Pp;

dis1 = x(1);
dis3 = x(2);
dis10 = x(3);
dis12 = x(4);

```

Figure 7.3: Main Kalman Filter code.

Figure 7.4 shows the main code which reads the real measurements from four sonars and passes them to SimpleKalmanFourState() to find the estimate of each of them. After receiving the estimate of the measurements, it presents them in four separate graphs.

```

ResultKalman4
clear all
Samples = 500;
X = zeros(Samples, 4);
Z1 = zeros(Samples, 1);
Z2 = zeros(Samples, 1);
Z3 = zeros(Samples, 1);
Z4 = zeros(Samples, 1);
for k=1:Samples
    z1 = GetMaxSonar1();
    z2 = GetMaxSonar3();
    z3 = GetMaxSonar10();
    z4 = GetMaxSonar12();

    [dis1, dis3, dis10, dis12] =
    KalmanFourStateV(z1,z2,z3,z4);

    X(k,:) = [dis1 dis3 dis10 dis12];
    Z1(k) = z1;
    Z2(k) = z2;
    Z3(k) = z3;
    Z4(k) = z4;
end
dt = 1;
t = 0:dt:Samples-dt;
figure
subplot(2,2,1)
plot(t, X(:,1),'r')
hold on
plot(t, Z1, 'k')
legend('Estimation S1','Measurements S1')
xlabel('Time(s)');
ylabel('Inche');
grid on
title('Sonar 1');

```

```

subplot(2,2,2)
plot(t, X(:,2),'y')
hold on
plot(t, Z2, 'k')
legend('Estimation S2','Measurements S2')
xlabel('Time(s)');
ylabel('Inche');
grid on
title('Sonar 3');
subplot(2,2,3)
plot(t, X(:,3),'g')
hold on
plot(t, Z3, 'k')
legend('Estimation S3','Measurements S3')
xlabel('Time(s)');
ylabel('Inche');
grid on
title('Sonar 10');
subplot(2,2,4)
plot(t, Z4, 'k')
hold on
plot(t, X(:,4),'b')
legend('Estimation S4','Measurements S4')
xlabel('Time(s)');
ylabel('Inche');
grid on
title('Sonar 12');

```

Figure 7.4: The code that Test Kalman Filter algorithm. This code passes four arguments which are the measurements coming from the sonars used in the wall following.

The results of the Kalman Filter estimations are shown in the Figure 7.5, Figure 7.6, Figure 7.7 and Figure 7.8. Kalman Filter gives a good estimation. It was able to avoid presenting the zero measurements in its estimation for all the sonars at the same time. However, there was a delay in the estimation, but this will not confuse the robot. In the first figure, the difference between the actual measurements were big, but the estimation of the real measurements were good. Even if it didn't match the regular measurements, it didn't

go to a level where it would confuse the robot. The estimation of sonar 3 and sonar 10 were closer to the “decision threshold”, which is the threshold beyond which the robot will make a decision. The decision can be halted if there is an obstacle less than ten inches ahead. Another decision might be to move away from the wall if the sonar on the right (sonar 10) measures a distance less than 15 inches. In the wall following algorithm, the front threshold is 10” and the side threshold is 15”. Nevertheless, the estimations were good and Kalman Filter eliminated most of the zeros.

Using Kalman Filter to estimate the sonars measurements was efficient and it prevented the unreal measurements (zeros) from causing delay to the wall following algorithm. It is clear that there exist differences between the measurements and their estimations, as shown in Figure 7.5, Figure 7.6, Figure 7.7 and Figure 7.8. This algorithm produced a faster response. Additionally, the robot has more confidence about its decisions. The disadvantages of using Kalman Filter is that there is a delay in estimating the right value, especially when there is a big change in measurements which can be seen in the estimation of Sonar 1.



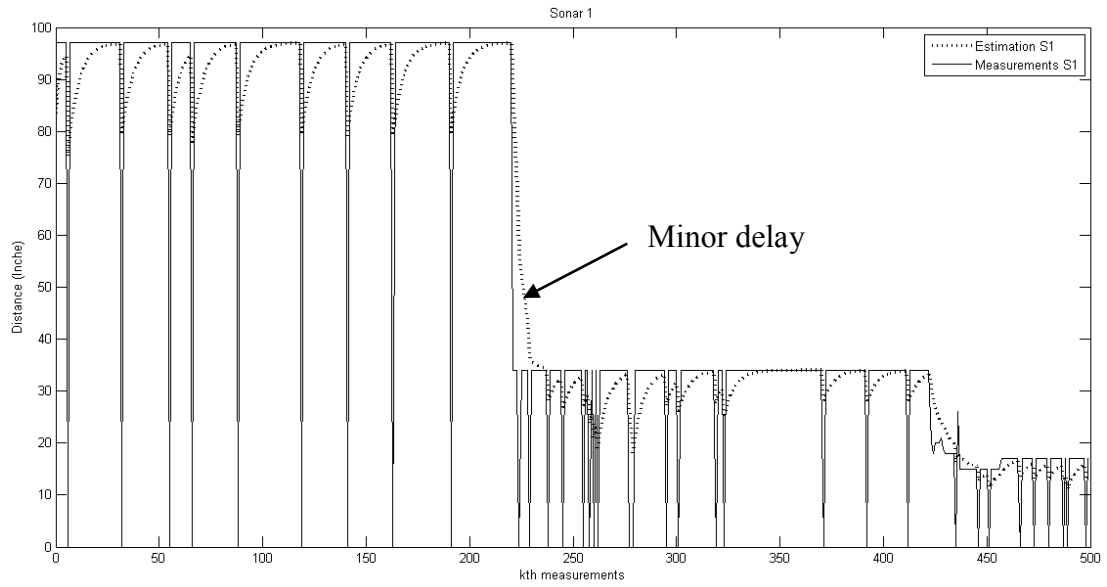


Figure 7.5: The zero removal algorithm refines the zero measurements from Sonar 1 using Kalman filter.

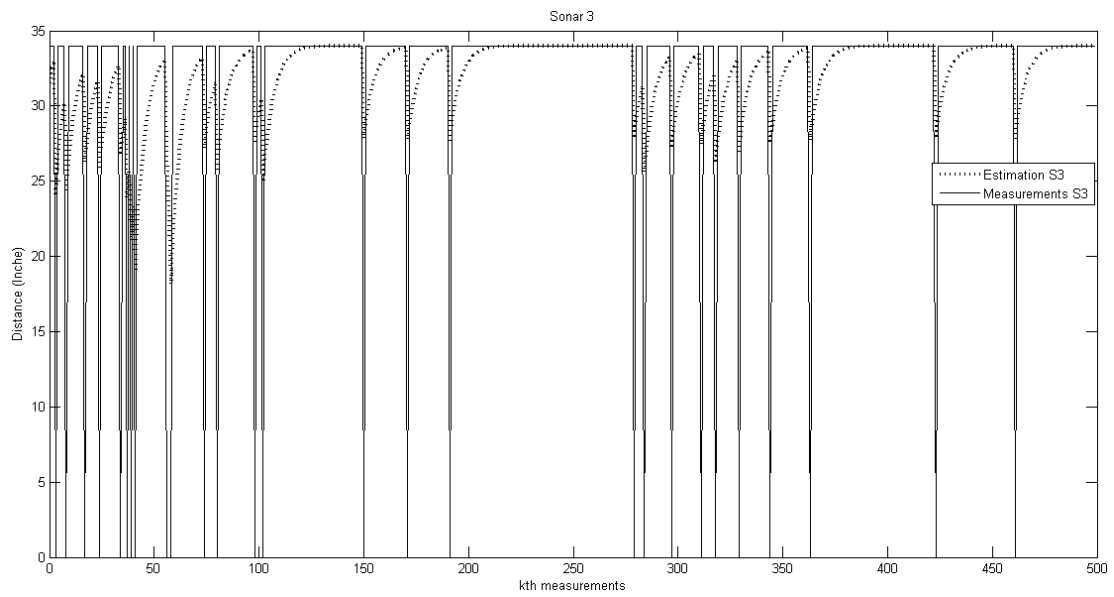


Figure 7.6: The zero removal algorithm refines the zero measurements with Kalman filter.

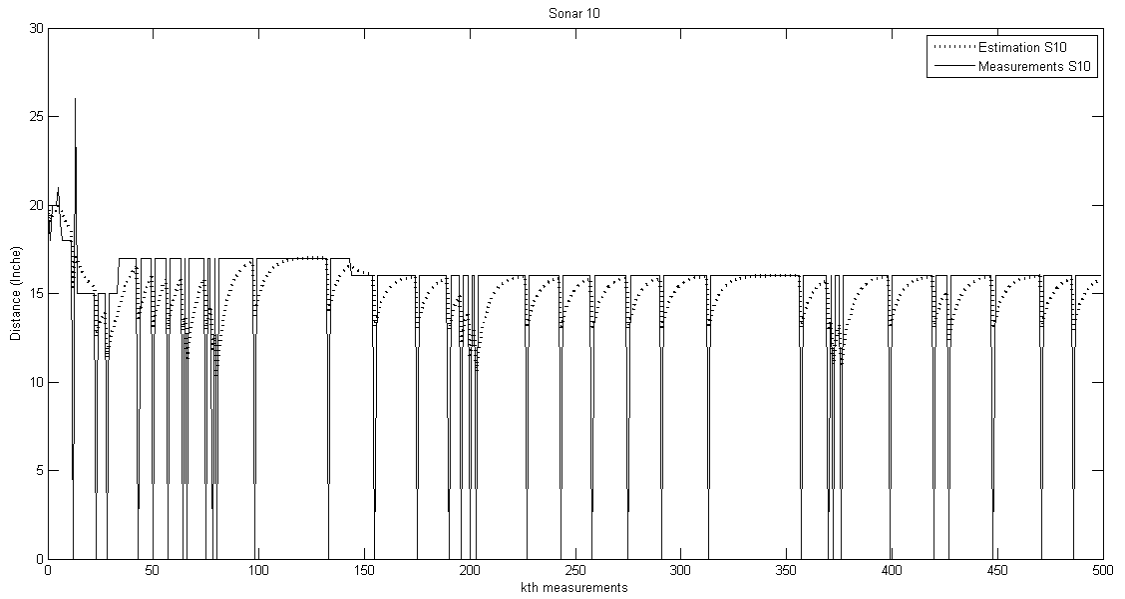


Figure 7.7: The zero removal algorithm refines the zero measurements with Kalman filter.

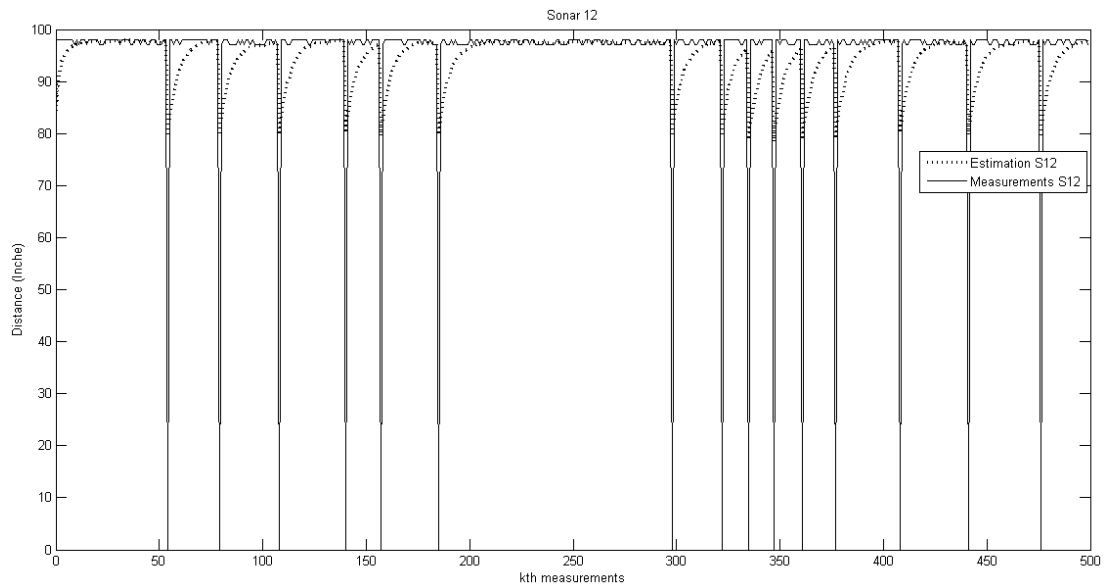


Figure 7.8: The zero removal algorithm refines the zero measurements with Kalman filter.

Using Kalman Filter here solves the problem of changing the wires with a shielded ones which would be more time consuming and also expensive. This would require many

hard hardware changes to the boards like soldering, change some connectors or move the boards to different place. Additionally, it might be that the problem will still exist even after make these hardware changes. For these reasons Kalman Filter was applied here.

## **7.2 Localization Based on Kalman Filter**

In this section, two different levels of Extended Kalman Filter are presented. The first one is solving the problem of localization with one-dimensional Extended Kalman Filter. While the second level will present two-dimensional Extended Kalman Filter to solve the problem of localization of the robot. Both of them will use sensor fusion. Sonars and Encoders will be fused in Extended Kalman Filter in both levels in order to get a better result than using sonars or encoders individually. Both one-dimensional and two-dimensional Extended Kalman Filter will be simulated using Matlab and tested on MCECS-Bot using the MRPT. The results of the tests will be presented as well as screen shots from the videos showing the real test are listed in the upcoming sections. A third level will be discussed theoretically. There will not be a real test for this level since implementing the Extended Kalman Filter for this level will be left as a future work.

### **7.2.1 One-Dimensional Extended Kalman Filter**

In this level, Extended Kalman Filter is tested to find the estimate of the position on one coordinate (either on x-axis or on y-axis). Which means, there is no simultaneous update for the position on x-axis and y-axis. Therefore, the robot updates the distance on

y-axis (if it starts moving on y-axis) then updates the position on x-axis (if it strafes right or left). In this case, the rotation is not implemented here. However, the robot is still able to move right and left due to the sliding ability of the Mecanum wheels. Encoders and the front sonars will be used to estimate the robot position away from a wall in a room. As presented previously, when the motors are running, at least one of the sonars gives a zero measurement (refer to ultrasonic section in chapter five). On the other hand, the encoders have the problem of accumulating error over time. This problem becomes bigger with Mecanum wheels due to slippage (refer to Mecanum wheels section in chapter five). The solution is fusing the sensors and encoders, which eliminates the zero measurements of sonars and the accumulated error of encoders as well as finds the best estimate of the position that came from these sensors. This solution will enhance the performance of the robot (Actually, there is no comparison between using Extended Kalman Filter and not using it!). Additionally, it will be cost-effective because we don't need to change the sonars, change the cables and/or add extra hardware parts to refine the signals.

The test is simply putting the MCECS-Bot in a room and make it face one of the walls. Before the robot starts to move, it measures the distance from the front wall then initializes this distance as the distance from the wall. The sonars (as explained in the fifth chapter) are reliable and work well when the motors are not running. This is what made the wall distance initializing possible. When the robot starts approaching the wall, it will update EKF with encoder and sonar measurements. EKF has two main models; state transition model and observation or measurements model. Encoders' measurements go to the state transition model, while sonars' measurements go to the observation model. Figure

7.9 shows the steps of the Extended Kalman Filter (for more information, review EKF section in chapter four).

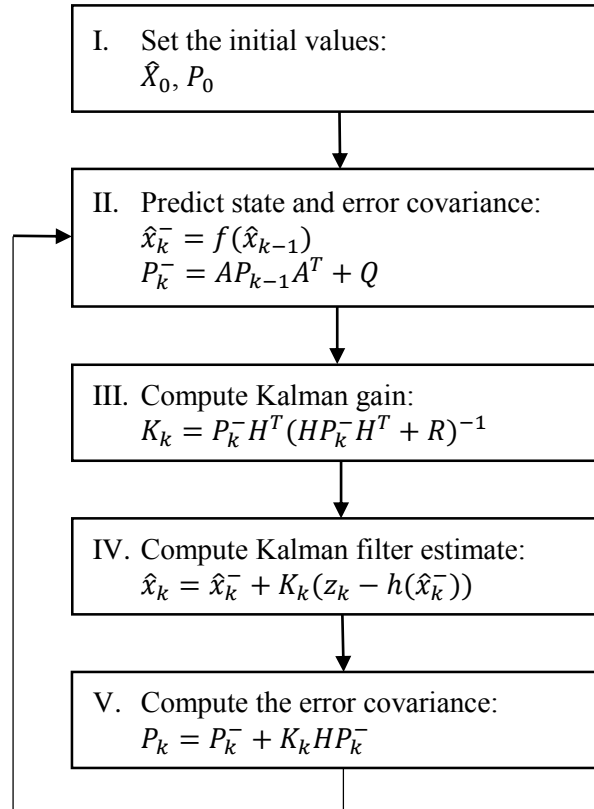


Figure 7.9: The steps of Extended Kalman Filter [12].

Figure 7.10 shows the poses of the robot in a room with the required measurements to apply the 1D EKF. As shown in this figure, the robot initializes the wall distance before it starts moving. Then the robot will find its position based on the measurements coming from encoders and sonars.

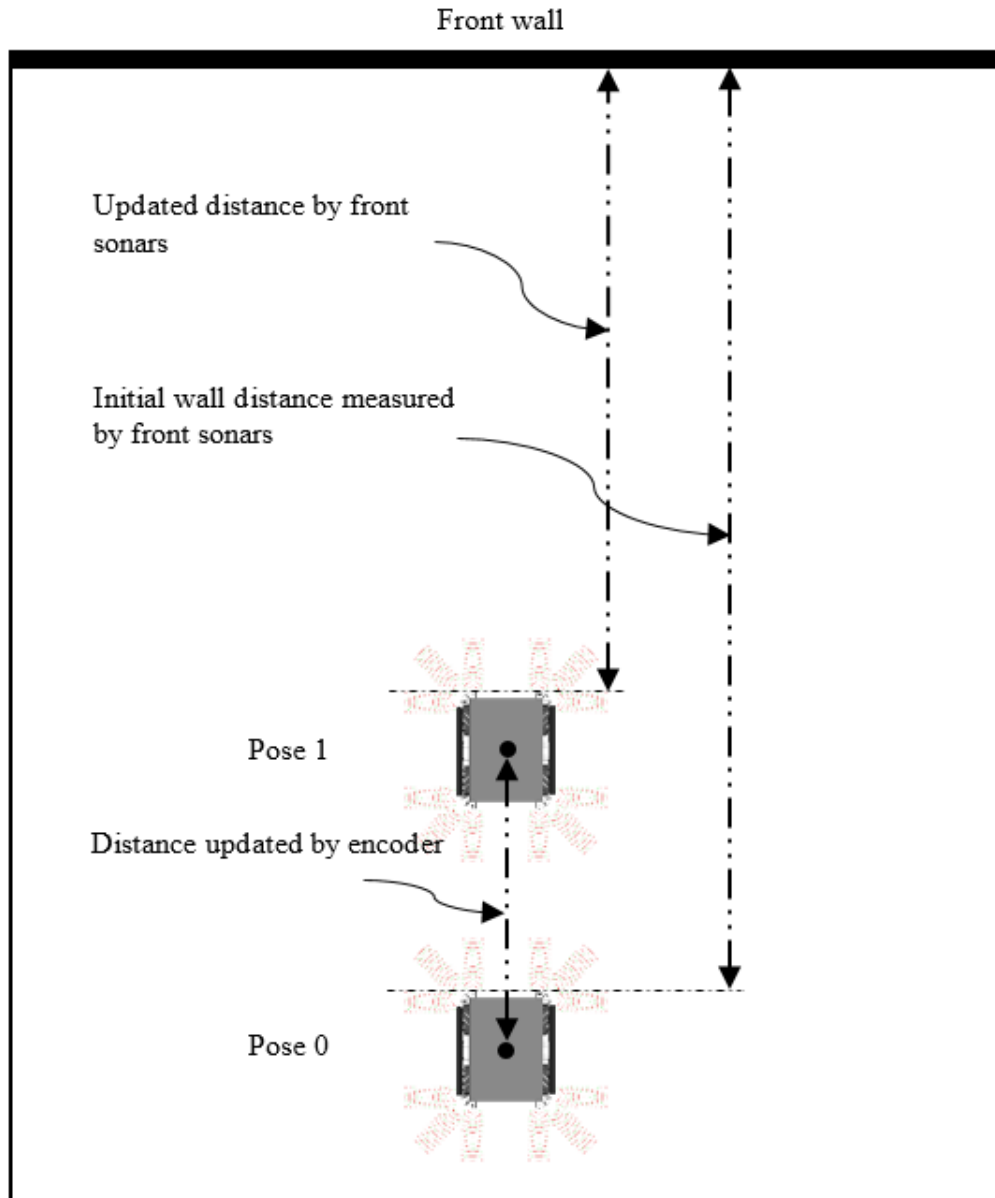


Figure 7.10: One dimensional Kalman Filter example.

As presented in Figure 7.9, the Kalman Filter consists of five steps that are iterated whenever there are measurements being fed to Extended Kalman Filter. In order to use the Extended Kalman Filter, we need to define what the state transient and observation model

of the system, the state transition mapping matrix ( $A$ ), and the measurements-state mapping matrix ( $H$ ) are. Let's first find the state space model equations of the system, which are:

$$x_{k+1} = f(x_k) + v_x$$

$$z_k = h(x_k) + w_x$$

In our case, the state of the system is the distance traveled on y-axis (or on x-axis). The current position equals the previous position plus the position measured by encoder as follows:

$$y_{k+1} = y_k + D_{Encoder} \quad \text{Equation 7.1}$$

The MRPT gets the updated position from Arduino Mega ( $x$ ,  $y$  and  $\theta$ ) away from the previous position. However, we need here only the additional distance traveled from the previous position ( $D_{Encoder}$  in equation 7.1). In this case, the MRPT code was edited to find the difference between the current position and the old one. Equation 7.1 can be written in different form:

$$y_{k+1} = y_k + \Delta t \cdot v_r, \text{ where } v_r \text{ is the robot speed.}$$

Anyway, we used the equation 7.1 to develop the Extended Kalman Filter. From this equation, the state transient equation of the robot was found:

$$y_{k+1} = y_k + D_{Encoder} + v_x \quad \text{Equation 7.2}$$

In order to calculate the mapping matrix  $A$ , we need to find the Jacobian of  $f(x)$ .

$$\frac{\partial f}{\partial x} = 1, \text{ which means } A = 1$$

The mapping matrix in equation 7.2 equals one because the state here describes the position of the robot on an axis which is not changing to another state. Value  $v_x$  is assumed here to be zero, because there is no error in describing the state of the robot while moving from one position to another.

There is a relationship between the initial wall distance, updated distance by the sonars and the state of the robot which depends on the old the state and the updated measurements from encoders (see figure 7.10). This relationship can be represented in the following equation:

$$S_{1,12} = D_{Wall} - y_k \quad \text{Equation 7.3}$$

Where  $S_{1,12}$  is the distance from the front wall measured by the average of sonar 1 and sonar 12. While  $D_{Wall}$  is the initial wall distance measured by the front sonar ( $S_{1,12}$ ) before the robot starts moving. Equation 7.3 is used to derive the observation model of the one-dimensional EKF. The observation model is:

$$S_{1,12_k} = D_{Wall} - y_k + w_k \quad \text{Equation 7.4}$$

Equation 7.4 tells us that the updated measurement of the front wall equals the difference between the initial wall distance and the state of the robot plus the error of the sonars. It is obvious that it is not possible to extract  $H$  matrix from equation 7.4. Therefore, the Jacobian of the  $h(x)$  must be found.

$$\frac{\partial h}{\partial x} = -1, \text{ which means } H = -1$$

$Q$  and  $R$  are found after experimenting with different values. These matrices have different effects on the estimate. When  $Q$  increases the weight of the measurements on the estimate



increases. The same thing happens when  $R$  decreases. Which means they have opposite influence on the behavior of the Extended Kalman Filter. So far, we found that  $A = 1$ ,  $H = -1$ . The best values for  $Q$  and  $R$  are 0.7 and 5.

### 7.2.1.1 Matlab Simulation of a One-Dimensional Extended Kalman Filter

In order to find out that my derivation and my representation for the MCECS-Bot system is correct, a simulation of the system together with EKF was needed. Therefore, encoders and sonars are simulated as well as the system state of MCECS-Bot. Additionally, to make the simulated model as close as possible to the real system, the accumulated error of the encoders was added to the encoder's measurements. On the other side, the zero measurements and noise of the sonars are added to their measurements as well. Figure 7.11 shows the code that simulates encoder measurements. The same code is repeated but with no noise in order to simulate the ideal measurements of the encoders without noise for the purpose of comparison (as shown in Figure 7.12).

```
GetEncoder
function E = GetEncoder()
persistent first_Run w A0
if isempty(first_Run)
    A0 = 0;
    w = 0;
    first_Run = 1;
end
E = A0 + w;
A0 = E+1;
w = 0 + (2*rand(1,1));
```

Figure 7.11: The code that simulates the measurements of the encoders.

GetRealEncoder
<pre> function R = GetRealEncoder() % % persistent first_Run persistent A0 if isempty(first_Run)     A0 = 0;      first_Run = 1; end  R = A0; A0 = R+1; </pre>

Figure 7.12: The code that simulates the ideal measurements of the encoders.

Figure 7.13 and Figure 7.14 do the same thing but with sonars.

GetSonar12
<pre> function z1 = GetSonar12() % % persistent first_Run w persistent A_1 if isempty(first_Run)     A_1 = 50;     first_Run = 1;     w = 0; end z1 = A_1-w; A_1 = z1-1; w = randn(1,1); </pre>

Figure 7.13: The code that simulates the measurements of sonar 12.

```

GetRealSonar
function z1 = GetRealSonar()
%
%
persistent first_Run w
persistent A_1
if isempty(first_Run)
    A_1 = 50;
    first_Run = 1;
    w = 0;
end
z1 = A_1+w;
A_1 = z1-1;
w = 0;

```

Figure 7.14: The code that simulates the measurements of sonar 12 but without noise or zero measurements.

Figure 7.15 shows the code of the Extended Kalman Filter. It receives the measurements of the encoders and the sensors and returns the estimate.

```

EKalman_1D
function [D,Px] = EKalman_1D (S,Inc_Enc)
%
%
persistent A H Q R Wd
persistent x P
persistent firstRun
Wd = 50;
z = S;
if isempty(firstRun)
    A = 1;
    H = -1;

    Q = 0.7; % Increasing Q will make the estimate closer
%to the measurements. Conrary, deceasing Q %decreases
%the dependency on measurements/observations.
    R = 5; % If the measurements are meant to have more
%effect on the estimate, matrix R should be decreased.

    x = 0;

```

```

P = 1;

firstRun = 1;
end

xp = A*(x+Inc_Enc);
Pp = A*P*A' + Q;

K = Pp*H'*inv(H*Pp*H' + R);

x = xp + K*(z - (Wd-x));
P = Pp - K*H*Pp;
D = x;
Px = P;

```

Figure 7.15: Extended Kalman Filter code.

Figure 7.16 lists the main code that receives the measurements of the encoders and sonars from their simulators and sends them to the EKF. Then, the main code receives the EKF estimate and presents it with all other measurements in a one figure.

```

Result1DEKalman1D
clear all
dt = 0.2;
t = 0:dt:10;
Encoder = 0;
Samples = length(t);
X = zeros(Samples, 2);
ZS = zeros(Samples, 1);
ZE1 = zeros(Samples, 1);
ZE2 = zeros(Samples, 1);
ZRE = zeros(Samples, 1);
ZRS = zeros(Samples, 1);
for k=1:Samples
    S = GetSonar12();
    R_S = GetRealSonar();
    E = GetEncoder();
    R_E = GetRealEncoder();
    ZE1(k) = E;

```

```

if (k<2)
    E_k_1 = 0;
    S = 50;
end
if (k == 5 || k == 10 || k == 18 || k == 24 || k == 35 || k ==
40 || k == 45)
    S = 0;
end
if (k > 1)
    E_k_1 = ZE1(k-1);
end
Inc_Enc = E - E_k_1;
Encoder = Encoder + Inc_Enc;
[D,Px] = Simple1DEKalman1D(S,Inc_Enc);
X(k,:) = [D Px];
ZS(k) = S;
ZE2(k) = E_k_1;
ZE1(k) = Encoder;
ZRE(k) = R_E;
ZRS(k) = R_S;
if (ZS(k)<0)
    ZS(k) = 0;
end
end
figure
plot(t, ZE1, 'k--','LineWidth',2)
hold on
plot(t, ZRE, 'k+')
plot(t, ZS, 'k:', 'LineWidth',2)
plot(t, ZRS, 'kx')
plot(t, X(:,1), 'ko','MarkerSize',10)
title('Kalman Filter Estimation');
xlabel('Time(s)');
ylabel('Inche');
xlabel('x-axis');
ylabel('y-axis');
legend('Encode measurements with noise','Real Encode
measurements without noise','Sonar measurements with
noise','Real Sonar measurements without noise','kalman
Filter Estimate')
grid on

```

Figure 7.16: The main code of the one-dimensional EKF simulator.

Figure 7.17 shows the results of the Extended Kalman Filter estimates. It clearly shows the “zero measurements problem” of the sonars as well as the accumulated error of the encoders.

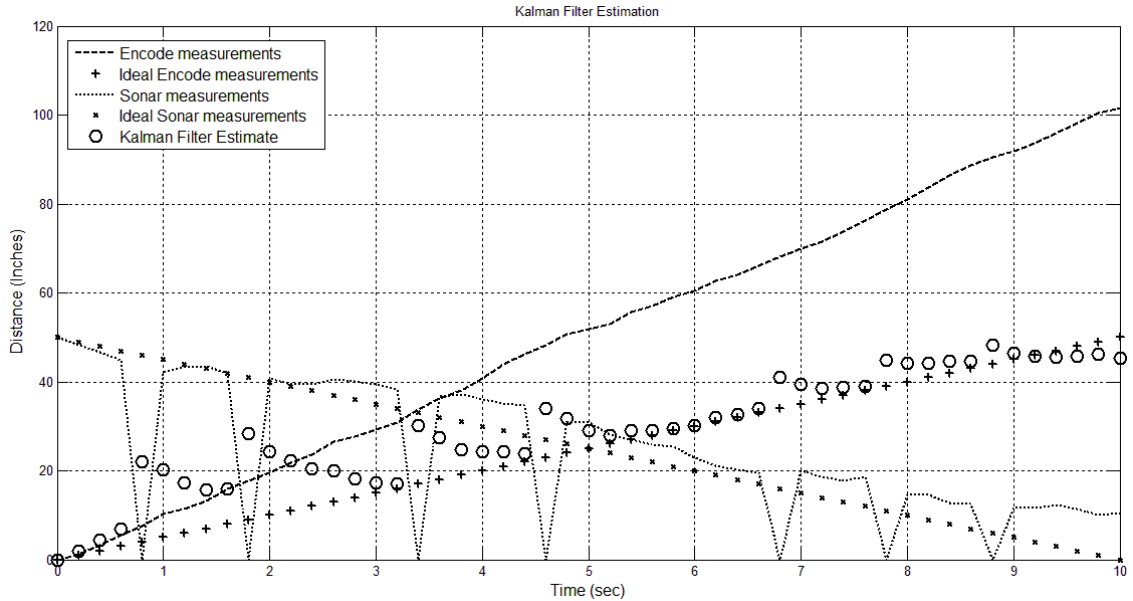


Figure 7.17: One-dimensional EKF result.

The goal of using Extended Kalman Filter is achieved here. It is clear that EKF eliminates the zero measurements of the sonars and the accumulated error of the encoders.

### 7.2.1.2 MRPT Results based on One-Dimension Extended Kalman Filter

MRPT has a built-in Extended Kalman Filter. In other words, the steps shown in Figure 7.9 are implemented and ready to use in MRPT. However, the equations that describe the system model and all required variables are needed. In a detailed explanation, the MRPT needs equations, matrices and their dimensions in order to process them with

the built in EKF. Therefore, we had to update the code of MRPT with all these new modules. The state transition equation model and the observation model were already derived in the previous section. The information fed to EKF in MRPT are:

- 1- State transition model:  $y_{k+1} = y_k + D_{Encoder} + v_x$ .
- 2- Observation model:  $S_{1,12}_k = D_{Wall} - y_k + w_k$ .
- 3- System matrices:  $A = 1, H = -1, Q = 0.7, R = 5$ .

One of the MRPT features is adding a map for the robot environment in order to show the robot position on the map. Moreover, it will show the sonars' beams (or cone) to show the robot sensing of its surrounding environment. The MRPT can show other objects and present them on the map, for instance; it shows the position of the robot based on EKF's estimate, sonars or odometry. In my experiment, the robot was mounted over a small cart (The wheels weren't in contact with the ground). There are dividers (represent walls) used to simulate the robot movements. When the wheels starts to spin, I push the wall against the robot so it can update its sonar measurements (As shown in Figure 7.18). This test is more secure than putting the robot on the ground and it is easier to collect the sensor data from the robot.



Figure 7.18: Running the test of the EKF.

The same equations and models described by the previous section are applied here in the MRPT code. To review MRPT code please refer to Appendix I. When the MRPT program starts running, it will show a map for the basement floor. Four objects will be shown on the map right after the map appears. The first object, which is a yellow ball, represents the position of the robot based on the measurements coming from encoder. The second object, which is a blue ball, represents the position of the robot based on the sonars' measurements. The third object, which is a green ball, represents the initial position of the robot. The last object, which is a small red cart, represents the robot's position based on the EKF estimates. There are only four sonars presented in these results. They were presented as a cone shape in front of the robot (Figure 7.19 shows the sonars' cone). It is possible to use all the twelve sonars but in our experiment only sonar1, sonar3, sonar10



and sonar 12 were used. The reason why we presented only these sonars because I didn't want them to be confused with other sonars if something would block them. I focused only on the sonars involved in the EKF algorithm at that moment.

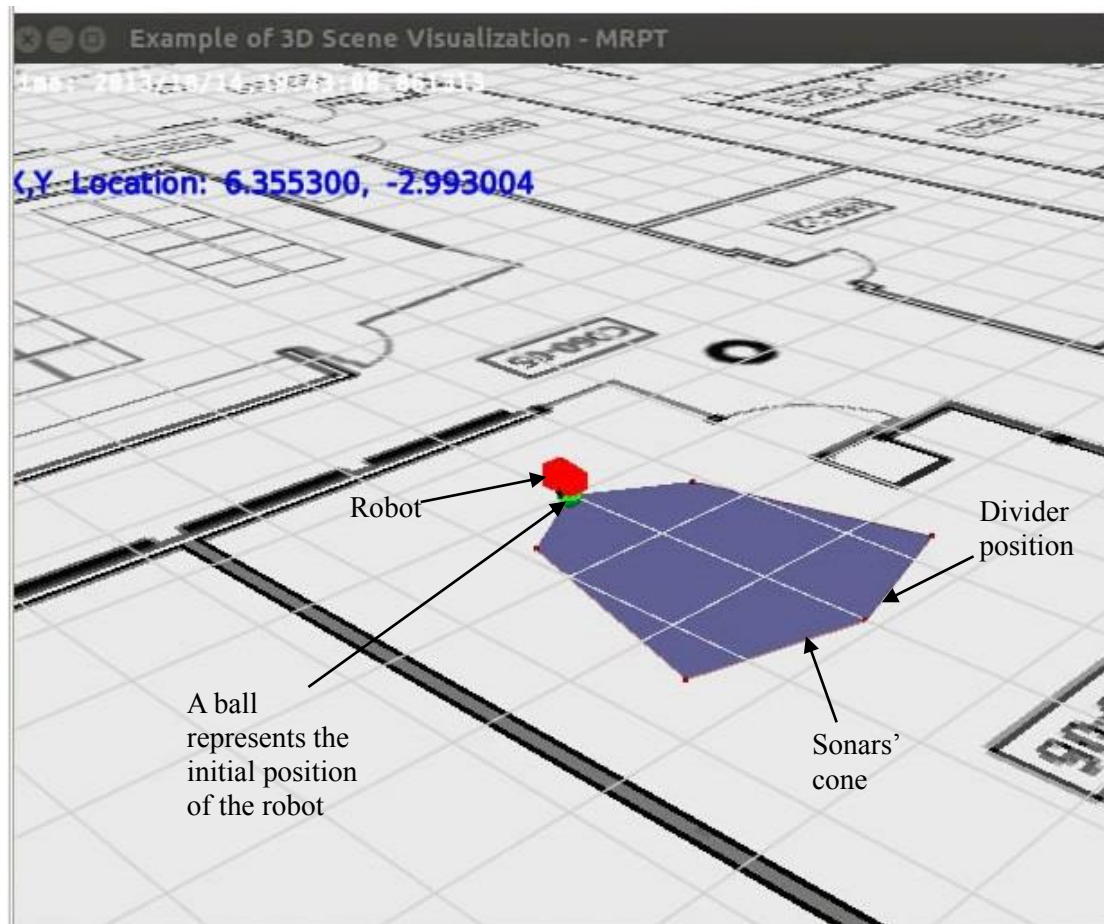


Figure 7.19: This window is produced by MRPT which is the map of the basement floor of the fab building.

On the program terminal, the data related to each object are presented. Figure 7.20 shows the MRPT program results that were presented on the terminal window that runs the MRPT code.

```
mcecsbot@jeeves: ~/MainProgram2/src/navigation/nav_main
mcecsbot@jeeves: ~/MainProgram2/... x mcecsbot@jeeves: ~ x
2.865
EKF: [0]
obs_2d IS NOT NULL!
*****Going Forward*****
[KF] 0 LMs | Pr: 0.00ms | Pr.Obs: 0.00ms | Obs.DA: 0.00ms | Upd: 0
.08ms
y=0.000, front_sonar=2.865, dy=0.000, initial front wall distance=
2.865
EKF: [0]
obs_2d IS NOT NULL!
*****Going Forward*****
[KF] 0 LMs | Pr: 0.00ms | Pr.Obs: 0.01ms | Obs.DA: 0.00ms | Upd: 0
.08ms
y=0.000, front_sonar=2.865, dy=0.000, initial front wall distance=
2.865
EKF: [0]
obs_2d IS NOT NULL!
*****Going Forward*****
[KF] 0 LMs | Pr: 0.01ms | Pr.Obs: 0.01ms | Obs.DA: 0.00ms | Upd: 0
.08ms
y=0.000, front_sonar=2.865, dy=0.000, initial front wall distance=
2.865
EKF: [0]
obs_2d IS NOT NULL!
7 odometry bytes got read...
Buffer has
*
*****Going Forward*****
[KF] 0 LMs | Pr: 0.01ms | Pr.Obs: 0.01ms | Obs.DA: 0.00ms | Upd: 0
.10ms
y=0.000, front_sonar=2.865, dy=0.000, initial front wall distance=
2.865
EKF: [0]
obs_2d IS NOT NULL!
*****Going Forward*****
[KF] 0 LMs | Pr: 0.00ms | Pr.Obs: 0.01ms | Obs.DA: 0.00ms | Upd: 0
.09ms
y=0.000, front_sonar=2.865, dy=0.000, initial front wall distance=
2.865
EKF: [0]
obs_2d IS NOT NULL!
read
x = 0, y = 1, phi = 359
```

Figure 7.20: This is the terminal window that runs the MRPT code. As shown in this figure, the information related to the robot position based on the sonars and encoders as well as the EKF estimates are presented in this terminal. The EKF estimate is zero here because the robot didn't start moving yet when this screen shoot was taken. The variable `front_sonar = 2.865` and the variable `initial front wall distance = 2.865` which means the robot is at the beginning of its path.

The result of the 1D EKF is presented on Figure 7.21, Figure 7.22 and Figure 7.23.

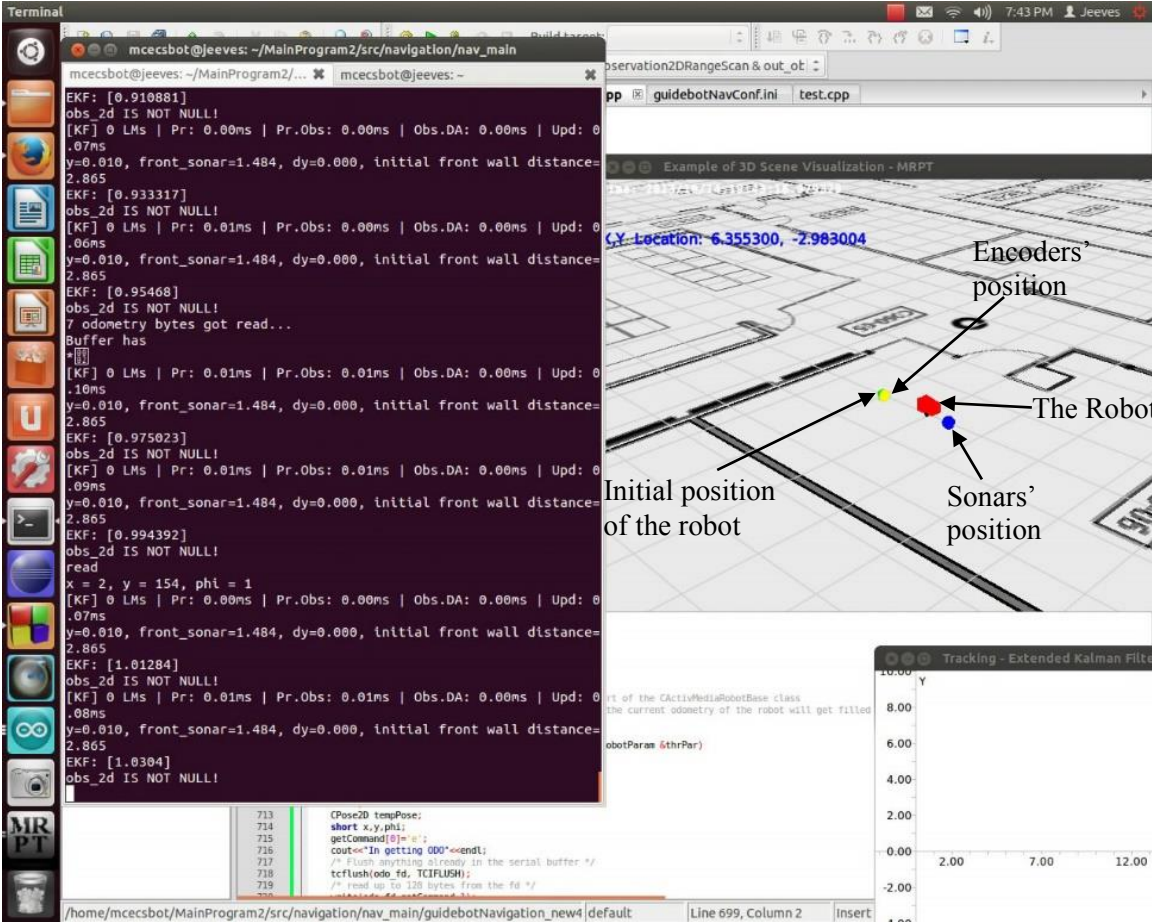


Figure 7.21: The biggest object, which represents the robot, is produced by EKF. It follows the ball that represents the robot position calculated based on the sonars' measurements. the two balls behind the robot are the initial position of the robot and the position of the robot based on the encoder's measurements. The window on the left reviews the information related to these objects.



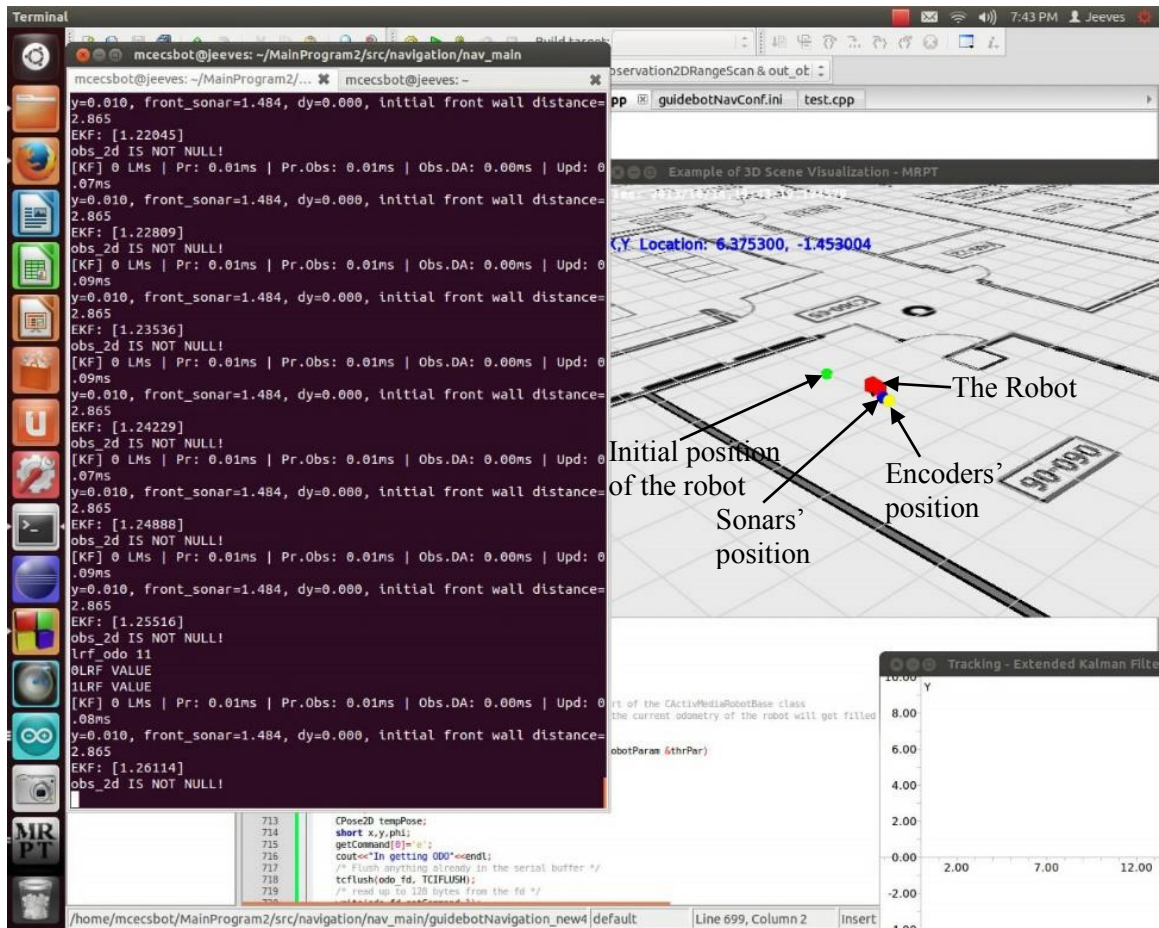


Figure 7.22: The first ball on the right (or the one that precedes the robot) is the robot's position based on the encoder's measurements. The ball that is behind the first ball is the position calculated based on the sonars' measurements. It is obvious that the robot is affected by the sensors more than encoders. The EKF neglected the encoder accumulated error. The window on the left reviews the information related to this process.

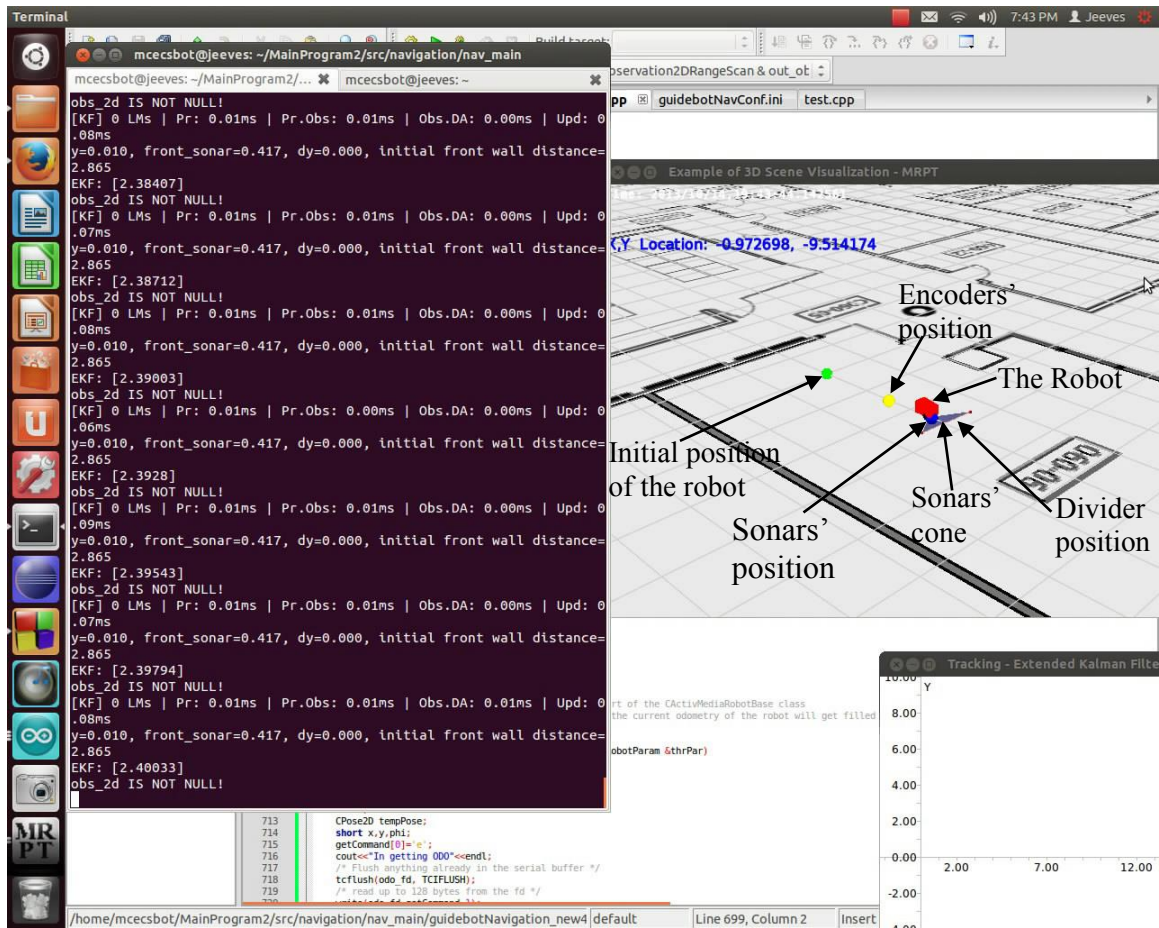


Figure 7.23: The final position of the robot after it stopped moving. The robot position, which is based on the EKF estimates, is between the sonars' based position and the encoders' based position. It is closer to the Sonars because it has to eliminate the accumulated error of the encoders. The cone here is small because the robot is very close to the wall (the divider shown in Figure 7.18).

The data shown in Figure 7.20 were printed to a CSV<sup>13</sup> file in order to draw them using Matlab. Consequently, the results can be seen more clearly in Figure 7.24.

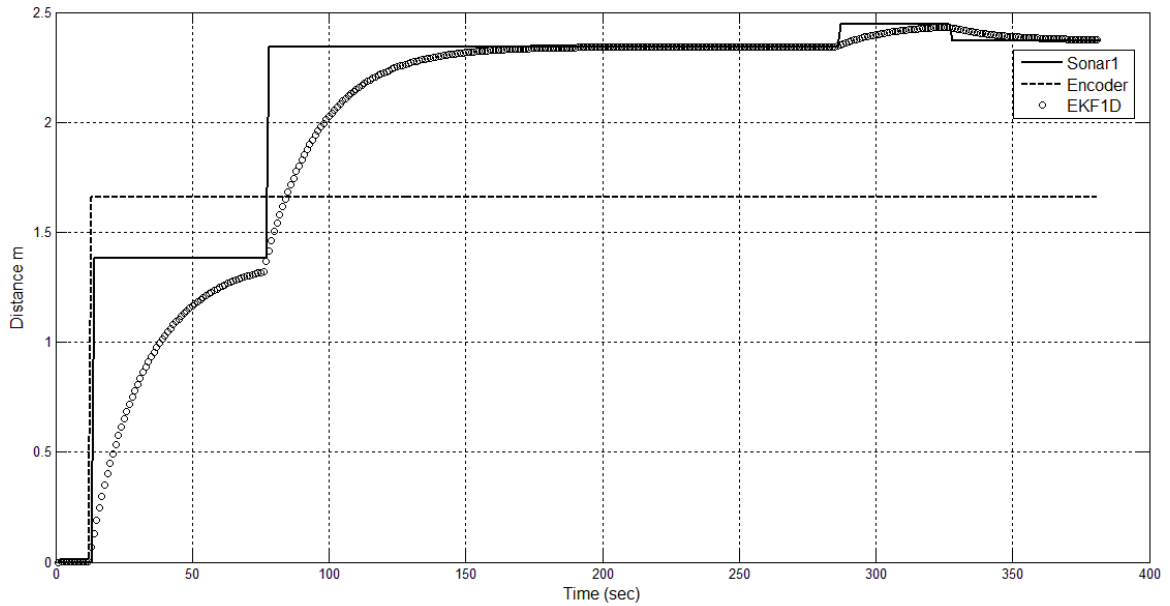


Figure 7.24: The result of using 1D EKF. The sonars measurements were good. There was an accumulated error coming from encoders. The estimates are more affected by sonars than by encoders.

---

<sup>13</sup> CSV stands for Coma Separated Values. This file is easier to read with matlab in order to present each data individually.

## 7.2.2 Two-Dimensional Extended Kalman Filter

In this level, EKF is used to find the best estimates of robot's position on x-axis and y-axis simultaneously. Therefore, the state of the robot consists of  $x$  and  $y$ . Here, a simulated and a real test are presented. Adding the simulated results makes understanding the system and its behavior easier. Additionally, Matlab code is much easier and less complicated than the MRPT code. The reader can easily follow the code and understand the EKF algorithm. The Matlab code of EKF is about 100 lines while MRPT code has more than 2500 lines! The simulation and real results are for the same system and for the same experiment. The experiment is putting the robot in a room and trying to find the estimate of the robot position based on the odometry and the sonars (which are front and right sonars or left and back sonars) measurements. Figure 7.25 shows the pose of the robot in a room. The solution is similar to the solution of the one-dimensional EKF. Simply, if the robot is about to move forward the distance from the front sonars should be initialized. Later, if it is about to move right/left the distance from the right/left wall should be initialized. The new updated measurements from the front/rear or right/left sonars equals the subsequent differences between the initial wall distance and the state of the robot which depends on the previous state and the distances updated based on odometry.

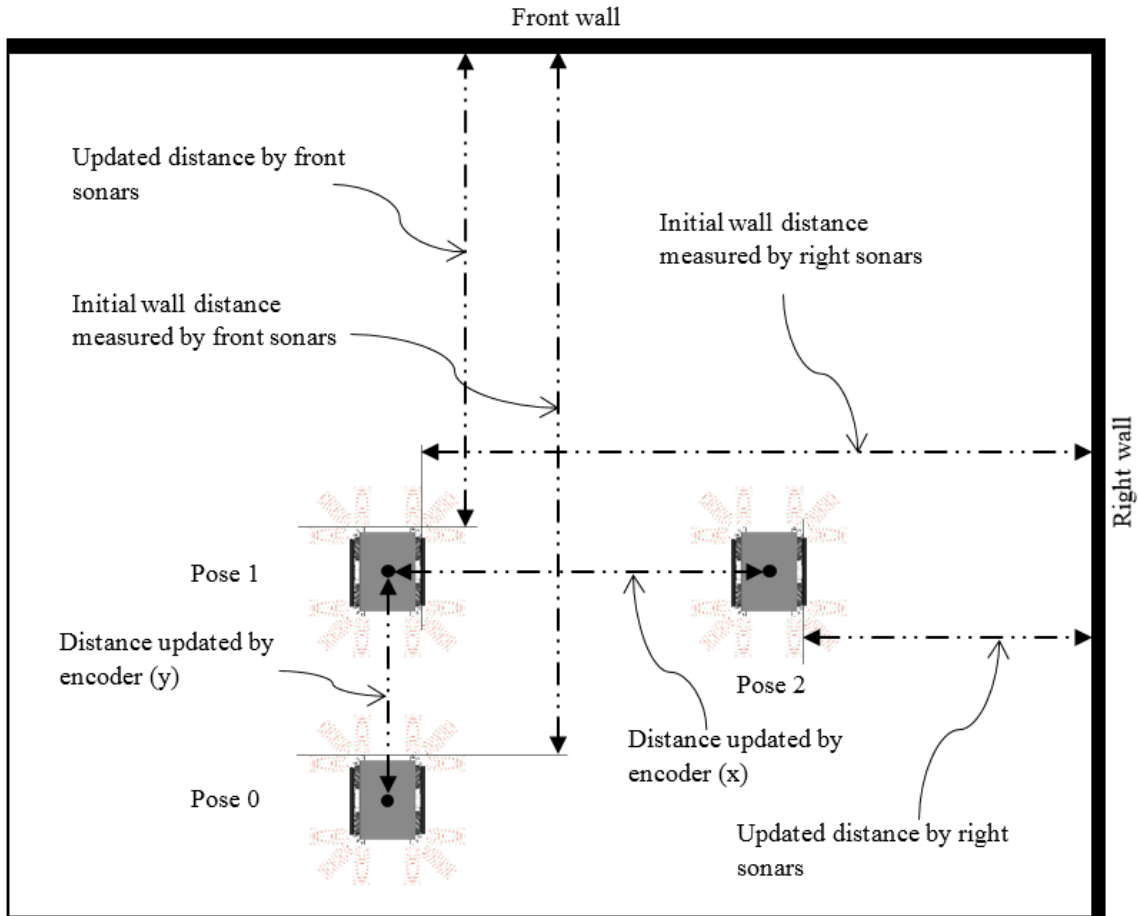


Figure 7.25: The robot's different poses while moving from y-axis to x-axis.

The same steps taken in the previous section that related to deriving the EKF equation will be followed here. The equations that describe the system state are:

$$x_{k+1} = x_k + D_{x\_Encoder} \quad \text{Equation 7.5}$$

$$y_{k+1} = y_k + D_{y\_Encoder} \quad \text{Equation 7.6}$$

As explained before, the encoder measurements coming from Arduino are updated based on the distance from the initial position. In equation 7.5 and equation 7.6, the



difference between previous position and current position is needed. These two equations lead us to find the state transition model which is shown in the equation below;

$$\begin{bmatrix} x \\ y \end{bmatrix}_k = \begin{bmatrix} x_k + D_{x\_Encoder} \\ y_k + D_{y\_Encoder} \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix} \quad \text{Equation 7.7}$$

$v_x$  and  $v_y$  in equation 7.7 are assumed to be zero because there is no error while transiting from state to state. It is not possible to find the  $A$  matrix from the equation above. Therefore, the Jacobian of the state transition equation is needed to find  $A$ , which is shown below:

$$A = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The relationship between the encoders and sonars can be listed in the following equations:

$$S9,10 = D_{x\_Wall} - x_k \quad \text{Equation 7.8}$$

$$S1,12 = D_{y\_Wall} - y_k \quad \text{Equation 7.9}$$

$S9,10$  is the average of the measurements coming sonar 9 and sonar 10 which updates the distance of the right wall (or left wall in case that sonar 3 and sonar 4 are used). On the other side,  $S1,12$  is average of the measurements coming from sonar 1 and sonar 12 which updates the distance from the front wall (or back wall in case sonar 6 and sonar 7 are used).  $D_{x\_Wall}$  and  $D_{y\_Wall}$  are the initial distance of the right and front walls based on the front and right sonars before the robot started to move. The Equations 7.8 and 7.9 lead to find the observation model of MCECS-Bot, which is shown below:

$$\begin{bmatrix} S9,10 \\ S1,12 \end{bmatrix}_k = \begin{bmatrix} D_{x\_Wall} - x_k \\ D_{y\_Wall} - y_k \end{bmatrix} + \begin{bmatrix} w_x \\ w_y \end{bmatrix} \quad \text{Equation 7.10}$$

$w_x$  and  $w_y$  are the errors of sonars measurements. Similar to  $A$ , the value of matrix  $H$  has to be calculated using the Jacobian method as follows:

$$H = \begin{bmatrix} \frac{\partial h_1}{\partial x} & \frac{\partial h_1}{\partial y} \\ \frac{\partial h_2}{\partial x} & \frac{\partial h_2}{\partial y} \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

Sensor fusion is also applied in this problem for the same reasons as explained in the one-dimensional EKF section. The reasons for using the EKF is to eliminate the zero measurements of the sonars and the accumulated error of the encoders. The last piece in EKF is to find  $Q$  and  $R$  matrices. As explained before, one way to find the best values of these matrices is by experimenting with different values and observing their effects on the system's response. In the two-dimensional EKF, the best values were the following:

$$Q = \begin{bmatrix} 14 & 0 \\ 0 & 14 \end{bmatrix}$$

$$H = \begin{bmatrix} 12 & 0 \\ 0 & 12 \end{bmatrix}$$

### 7.2.2.1 Two-Dimensional Extended Kalman Filter Matlab Simulation

The same reasons of simulating the 1D EKF on Matlab, the 2D EKF is simulated here. In Figure 7.26 and Figure 7.27 the codes that are responsible for generating the encoder measurements on the x-axis and y-axis are presented. The code simulates the accumulated error of the encoder. In Figure 7.28 and Figure 7.29 the software codes that tell us the ideal x and y positions, respectively, are presented. We can use these ideal values to compare them with the odometry results.

```
GetEncoder_X
function z2 = GetEncoder_X(b)
persistent A_1 A_2 A_3 A_4 A_5 C A_6 B A_7 w
if (b<11)
    if (b==1)
        A_1 = 0;
        w = 0;
    end
    z2 = A_1+w;
    w = rand(1,1);
    A_2 = z2;
elseif (b>10 && b<21)
    w = 2* rand(1,1);
    z2 = A_2+w;
    A_2 = z2+1;
    A_3 = z2;
elseif (b > 20 && b < 40)
    w = rand(1,1);
    z2 = A_3 + w;
    A_4 = z2;
elseif (b>39 && b<55)
    w = 2* rand(1,1);
    z2 = A_4+w;
    A_4 = z2 + 1;
    A_5 = z2;
elseif (b>54 && b< 75)
    w = rand(1,1);
    z2 = A_5 + w;
    A_6 = z2;
```

```

elseif (b>74 && b<85)
    w = 2*rand(1,1);
    z2 = A_6+w;
    A_6 = z2 +1.6;
    A_7 = z2;
else
    w = rand(1,1);
    z2 = A_7+w;
end

```

Figure 7.26: The code that simulates the odometry measurements on x-axis. The measurement variable *b* (in the code) is passed by the main code. It represents the order of the measurement. It is used to generate different area of measurements. As seen later in EKF results, the encoder results will diverge from the real ones. Additionally, this code generates an error that is added to the new measurements. This error represented by *w*.

```

GetEncoder_Y
function z1 = GetEncoder_Y(b)
persistent w
persistent A_1 A_2 A_3 A_4 A_5 C B A_6 A_7 A_8
if (b<11)
    if (b==1)
        A_1 = 0;
        w = 0;
    end
    z1 = A_1+w;
    A_1 = z1+1;
    w = rand(1,1);
    A_2 = z1;
    % B = 0;
elseif(b>10 && b<21)
    w = rand(1,1);
    z1 = A_2+w;
    A_3 = z1;
    % B = B+0.2;
elseif (b > 20 && b < 40)
    w = rand(1,1);
    z1 = A_3 + w;
    A_3 = z1 + 1;
    A_4 = z1;
    % C = 1;
elseif (b>39 && b<55)
    w = rand(1,1);

```

```

z1 = A_4+w;
A_5 = z1;
% C = C+0.2;
elseif (b>54 && b< 65)
    w = rand(1,1);
    z1 = A_5 + w;
    A_5 = z1 + 3;
    A_6 = z1+1.8;
elseif (b>64 && b<75)
    w = rand(1,1);
    z1 = A_6 + w;
    A_6 = z1 + 2.7;
    A_7 = z1+1;
    B= 0;
elseif (b>74 && b<85)
    w = rand(1,1);
    z1 = A_7 + w;% + B;
% B= B + 1;
    A_8 = z1;
else
    w = rand(1,1);
    z1 = A_8+w;%B;
    B= B + 1;
end

```

Figure 7.27: The code that simulates the odometry measurements on y-axis. Variable b and w work in similar way as in GetEncoder\_X.

```

GetRealX
function z2 = GetRealX(b)
%
%
persistent w

persistent A_1 A_2 A_3 A_4 A_5 C A_6 B A_7
if (b<11)
    if (b==1)
        A_1 = 0;
        w = 0;
    end
z2 = A_1+w;
%A_1 = z2-1;
w = 0;

```

```

A_2 = z2;
elseif(b>10 && b<21)
    w = 0;
    z2 = A_2-w;
    A_2 = z2+1;
    A_3 = z2;
    % B=0.5;
elseif (b > 20 && b < 40)
    w = 0;
    z2 = A_3 + w;
    A_4 = z2;
    % B = B+0.5;
    % C =0.5;
elseif (b>39 && b<55)
    w = 0;
    z2 = A_4+w;
    A_4 = z2 + 1;
    A_5 = z2;
elseif (b>54 && b< 75)
    w = 0;
    z2 = A_5 + w;
    A_6 = z2;
    % C = C + 0.5;
elseif (b>74 && b<85)
    w = 0;
    z2 = A_6+w;
    A_6 = z2 +1.6;
    A_7 = z2;
else
    w = 0;
    z2 = 40+w;
    % A_7 = z2+0.4;
end

```

Figure 7.28: The code that simulates the real position of the robot on x-axis. Variable b works in similar way as it does in GetEncoder\_X. Variable w here equals zero as there is no error.

GetRealY
<pre> function z1 = GetRealY(b) persistent A_1 A_2 A_3 A_4 A_5 C B A_6 A_7 w if (b&lt;11)     if (b==1)         A_1 = 0;         w = 0;     end     z1 = A_1+w;     A_1 = z1+1;     w = 0;     A_2 = z1; elseif(b&gt;10 &amp;&amp; b&lt;21)     w = 0;     z1 = A_2+w;     A_3 = z1; elseif (b &gt; 20 &amp;&amp; b &lt; 40)     w = 0;     z1 = A_3 + w;     A_3 = z1 + 1;     A_4 = z1; elseif (b&gt;39 &amp;&amp; b&lt;55)     w = 0;     z1 = A_4+w;     A_5 = z1; elseif (b&gt;54 &amp;&amp; b&lt; 65)     w = 0;     z1 = A_5 + w;     A_5 = z1 + 3;     A_6 = z1+1.8; elseif (b&gt;64 &amp;&amp; b&lt;75)     w = 0;     z1 = A_6 + w;     A_6 = z1 + 2.7;     A_7 = z1+1; elseif (b&gt;74 &amp;&amp; b&lt;85)     w = 0;     z1 = 80 + w;% + B; else     w = 0;     z1 = 80+w; end </pre>

Figure 7.29: The code that simulates the real position of the robot on y-axis. Variable b works in similar way as it does in GetEncoder\_X. w here equals zero as there is no error.

Figure 7.30 and Figure 7.31 show the codes that simulate the sonar measurements which are sonar1 (the front sonar) and sonar10 (the right sonar). The codes simulate these measurements with a zero measurements as well as with a noise to simulate a real sonar response. Figure 7.32 and Figure 7.33 show the codes that simulate the sonar measurements but without any noise or zero measurements. It is useful to have these measurements for comparison purpose.

```

GetSonar1
function z1 = GetSonar1(b)
persistent A_1 A_2 A_3 A_4 A_5 w
if (b<11)
    if (b==1)
        A_1 = 80;
        w = 0;
    end
    z1 = A_1+w;
    A_1 = z1-1;
    w = 1 * randn(1,1);
    A_2 = z1;
elseif(b>10 && b<21)
    w = 1 * randn(1,1);
    z1 = A_2+w;
    A_3 = z1;
elseif (b > 20 && b < 40)
    w = 1 * randn(1,1);
    z1 = A_3 + w;
    A_3 = z1 - 1;
    A_4 = z1;
elseif (b>39 && b<55)
    w = 1 * randn(1,1);
    z1 = A_4+w;
    A_4 = z1;
elseif (b>54 && b< 65)
    w = 1 * randn(1,1);
    z1 = A_4 + w;
    A_4 = z1 - 4;
    A_5 = z1;
elseif (b>64 && b<75)
    w = 1 * randn(1,1);

```



```

z1 = A_5 + w;
A_5 = z1 - 1.8;
else
w = 1 * randn(1,1);
z1 = 0+w;
end

```

Figure 7.30: The code that simulates sonar1 measurement which is the distance from the front wall. Variables b and w work in a similar way as in GetEncoder\_X.

```

GetSonar10
function z2 = GetSonar10(b)
persistent w

persistent A_1 A_2 A_3 A_4 A_5
if (b<11)
    if (b==1)
        A_1 = 40;
        w = 0;
    end
    z2 = A_1+w;
    %A_1 = z2-1;
    w = 0.5 * randn(1,1);
    A_2 = z2;
elseif(b>10 && b<21)
    w = 0.5 * randn(1,1);
    z2 = A_2+w;
    A_2 = z2-1;
    A_3 = z2;
elseif (b > 20 && b < 40)
    w = 0.5 * randn(1,1);
    z2 = A_3 + w;
    A_4 = z2;
elseif (b>39 && b<55)
    w = 0.5 * randn(1,1);
    z2 = A_3+w;
    A_3 = z2 - 1;
    A_4 = z2;
elseif (b>54 && b< 75)
    w = 0.5 * randn(1,1);
    z2 = A_4 + w;
    A_5 = z2;
elseif (b>74 && b<85)

```

```

w = 0.5 * randn(1,1);
z2 = A_5+w;
A_5 = z2 -1.9;
else
w = 0.5 * randn(1,1);
z2 = 0+w;
end

```

Figure 7.31: The code that simulates sonar10 measurement which is the distance from the right wall. b and w work in similar way as in GetEncoder\_X.

```

GetSonar1_Real
function z1 = GetSonar1_Real(b)
persistent w

persistent A_1 A_2 A_3 A_4 A_5
if (b<11)
    if (b==1)
        A_1 = 80;
        w = 0;
    end
    z1 = A_1+w;
    A_1 = z1-1;
    w = 0;
    A_2 = z1;
elseif(b>10 && b<21)
    w = 0;
    z1 = A_2+w;
    A_3 = z1;
elseif (b > 20 && b < 40)
    w = 0;
    z1 = A_3 + w;
    A_3 = z1 - 1;
    A_4 = z1;
elseif (b>39 && b<55)
    w = 0;
    z1 = A_4+w;
    A_4 = z1;
elseif (b>54 && b< 65)
    w = 0;
    z1 = A_4 + w;
    A_4 = z1 - 4;
    A_5 = z1;

```

```

elseif (b>64 && b<75)
    w = 0;
    z1 = A_5 + w;
    A_5 = z1 - 1.8;
else
    w = 0;
    z1 = 0+w;
end

```

Figure 7.32: The code that simulates the measurements of the front sonar without noise or zero measurements. Therefore, w equals zero here.

```

GetSonar10_Real
function z2 = GetSonar10_Real(b)
%
%
persistent w

persistent A_1 A_2 A_3 A_4 A_5
if (b<11)
    if (b==1)
        A_1 = 40;
        w = 0;
    end
    z2 = A_1+w;
    %A_1 = z2-1;
    w = 0;
    A_2 = z2;
elseif (b>10 && b<21)
    w = 0;
    z2 = A_2+w;
    A_2 = z2-1;
    A_3 = z2;
elseif (b > 20 && b < 40)
    w = 0;
    z2 = A_3 + w;
    A_4 = z2;
elseif (b>39 && b<55)
    w = 0;
    z2 = A_3+w;
    A_3 = z2 - 1;
    A_4 = z2;

```

```

elseif (b>54 && b< 75)
    w = 0;
    z2 = A_4 + w;
    A_5 = z2;
elseif (b>74 && b<85)
    w = 0;
    z2 = A_5+w;
    A_5 = z2 -1.9;
else
    w = 0;
    z2 = 0+w;
end

```

Figure 7.33: The code that simulates the measurements of the right sonar without noise or zero measurements. Therefore, w equals zero here.

Figure 7.34 shows the code of the 2D EKF. It receives the encoders' measurements (on x-axis and on y-axis) and the sonar data (the front sonar and the right sonar). Then, it return the estimate of the robot position on the (x, y) coordinate.

```

EKalman_2D
function [Dx, Dy] = EKalman_2D(SF, SR, Inc_Enc_x,
Inc_Enc_y)
persistent A H Q R Wd_x Wd_y
persistent x P
persistent firstRun
Wd_x = 40;
Wd_y = 80;
z = [SR SF]';
if isempty(firstRun)
    A = 1 * eye(2);
    H = [-1 0;
         0 -1];
    Q = 14 * eye(2); % Increasing Q will make the
estimate %closer to the measurements. Contrary,
decreasing Q %decreases the dependency on
measurements/observations.
    R = 12 * eye(2); % If the measurements are meant to
%have more effect on the estimate, matrix R should be
%decreased.

```

```

x = [0 0]';
P = 25 * eye(2);
firstRun = 1;
end
xp = A*(x+[Inc_Enc_x Inc_Enc_y]');
Pp = A*P*A' + Q;

K = Pp*H'*inv(H*Pp*H' + R);

x = xp + K*(z - ([Wd_x Wd_y]' - [x(1) x(2)]));
P = Pp - K*H*Pp;
Dx = x(1);
Dy = x(2);

```

Figure 7.34: Two-Dimensional Extended Kalman Filter code.

Figure 7.35 shows the main code that calls all the functions from the above figures, obtains the encoders' and sonars' measurements, and passes them as arguments to the EKalman\_2D. Then, it receives the estimate of the robot position. Eventually, it present the results of the 2D EKF with all the measurements from the encoders and sonars beside the measurements without noise.

```

Result2DEKalman
clear all
dt = 0.2;
t = 0:1:90;
Encoder = 0;
Samples = length(t);

X = zeros(Samples, 2);
ZS_F = zeros(Samples, 1);
Z2 = zeros(Samples, 1);
ZS_R = zeros(Samples, 1);
Z_Ex = zeros(Samples, 1);
Z_Ey = zeros(Samples, 1);
Z_Ex_k_1= zeros(Samples, 1);
Z_Ey_k_1= zeros(Samples, 1);
Z_Rx = zeros(Samples, 1);

```

```

Z_Ry = zeros(Samples, 1);
Z_RS_F = zeros(Samples, 1);
Z_RS_R = zeros(Samples, 1);
D_R = zeros(Samples, 2);
b = 0;
for k=1:Samples

    b = b+1;

    SR = GetSonar10(b);
    SF = GetSonar1(b);

    E_x = GetEncoder_X(b);
    E_y = GetEncoder_Y(b);
    R_x = GetRealX(b);
    R_y = GetRealY(b);

    Real_FS = GetSonar1_Real(b);
    Real_RS = GetSonar10_Real(b);

    Z_Ex(k) = E_x;
    Z_Ey(k) = E_y;

    if (k<2)
        E_k_1_x = 0;
        E_k_1_y = 0;
        SF = 80;
        SR = 40;
    end

    if (k > 1)
        E_k_1_x = Z_Ex(k-1);
        E_k_1_y = Z_Ey(k-1);
    end

    Inc_Enc_x = E_x - E_k_1_x;
    Inc_Enc_y = E_y - E_k_1_y;

    [Dx, Dy] = EKalman_2D(SF, SR, Inc_Enc_x,
Inc_Enc_y);

    if (k == 5 || k == 10 || k == 18 || k == 24 || k == 45 || k ==
50 || k == 65 || k == 77)

```

```

    SR = 0;
    %SF = 0;
end
if (k == 15 || k == 20 || k == 25 || k == 30 || k == 35 || k ==
55 || k == 70 || k == 80)
    %SR = 0;
    SF = 0;
end
ZS_F(k) = SF;
ZS_R(k) = SR;
Z_Rx(k) = R_x;
Z_Ry(k) = R_y;
Z_Ex_k_1(k) = E_k_1_x;
Z_Ey_k_1(k) = E_k_1_y;
D_R(k,:) = [Dx, Dy];
Z_RS_F(k) = Real_FS;
Z_RS_R(k) = Real_RS;

end
figure
plot(ZS_R, ZS_F, 'kx:', 'MarkerSize', 10)
hold on
plot(Z_RS_R, Z_RS_F, 'k.', 'LineWidth', 2)
plot(Z_Ex, Z_Ey, 'k+')
plot(Z_Rx, Z_Ry, 'k--', 'LineWidth', 2)
plot(D_R(:,1), D_R(:,2), 'ko-', 'MarkerSize', 10)

% figure
% plot(Z_Ex, Z_Ey, 'b+')
% hold on
% plot(Z_Rx, Z_Ry, 'k+')
% plot(t, Z_Rx, 'k+')
% plot(t, Z_Ey, 'bo')

legend('Sonar Measurements with noise', 'Real Sonar
Measurements without noise', 'Encoder Measurements with
noise', 'Real Encoder Measurements without noise', 'kalman
Filter Estimates')

xlabel('x-axis');
ylabel('y-axis');
grid on

```

Figure 7.35: The main code that coordinates the process of simulating 2D EKF.

The results of the 2D EKF is shown in Figure 7.36.

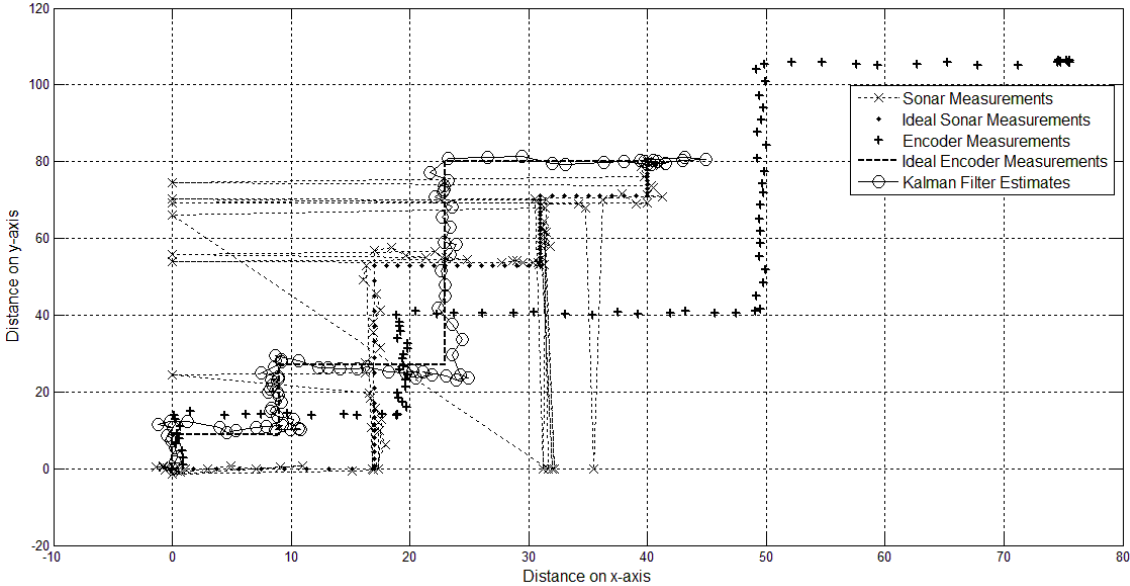


Figure 7.36: The result of the 2D EKF.

It is clear in the previous figure that the encoder measurements are off from the real position. The lines that go from the sonar measurements towards x and y axis represent the zero measurements of the sonars. The result shows us that Extended Kalman Filter did a good job with fusing these two sensors and combined their best performance (strengths). The result was an estimate without the zero measurements of the sonars and the accumulated error of the encoders.



### 7.2.2.2 Two-Dimensional Extended Kalman Filter MRPT Results<sup>14</sup>

The same procedures I did in section “7.2.1.2 One-Dimension Extended Kalman Filter MRPT Results” are followed here. The difference is that the system became more complicated and it is now a two-dimension instead of a one-dimension system. The equations derived in the previous section will be initialized in the MRPT code. The information fed to EKF in MRPT are:

1- State transition model:  $\begin{bmatrix} x \\ y \end{bmatrix}_k = \begin{bmatrix} x_k + D_{x\_Encoder} \\ y_k + D_{y\_Encoder} \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix}.$

2- Observation model:  $\begin{bmatrix} S_{9,10} \\ S_{1,12} \end{bmatrix}_k = \begin{bmatrix} D_{x\_Wall} - x_k \\ D_{y\_Wall} - y_k \end{bmatrix} + \begin{bmatrix} w_x \\ w_y \end{bmatrix}.$

3- System matrices:  $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, H = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, Q = \begin{bmatrix} 14 & 0 \\ 0 & 14 \end{bmatrix},$

$$R = \begin{bmatrix} 12 & 0 \\ 0 & 12 \end{bmatrix}.$$

The same way that the EKF 1D was tested in the 1D section is used to test the algorithm of 2D EKF. There are two dividers (moving walls) used to simulate the robot approach towards them. The robot first received a command to move forward then received another command to move right. The information was presented in the Linux terminal

---

<sup>14</sup> Unfortunately, one of the Motor controllers (the one that is controlling the rear wheels) stopped and the only way to fix it is to send it to the manufacturer. This will take long time and I have to submit the thesis as required on time. Fortunately, strafing sliding movements have the same speed as moving forward/backward in Mecanum wheels. Therefore, when the MRPT send the command to strafe left or right, the Arduino will control the front wheels to move forward and send the forward speed as a sliding speed. This is used in MRPT as sliding speed to be fed to EKF.

window when the MRPT code were run. Additionally, all the data were saved in a CSV file extension in order to represent the data to be used later by Matlab. The same objects appeared here, which are three balls representing the robot's initial position and the robot's position based on the encoders and the sonars, and the fourth object represents the robot's position based on the EKF estimates. The results are shown in Figure 7.37, Figure 7.38 Figure 7.39, Figure 7.40 and Figure 7.41.

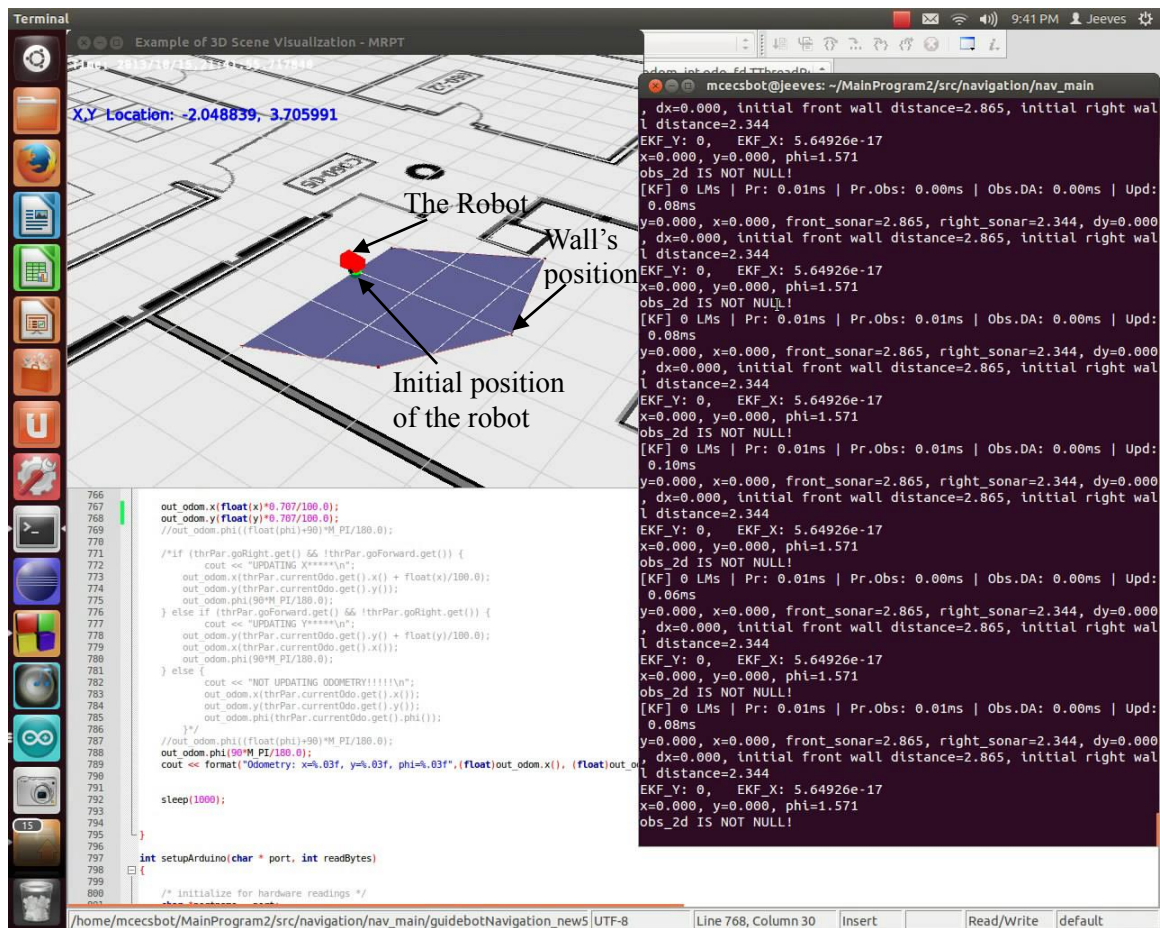


Figure 7.37: The robot is now stationary. Probably, it is taking measurements using sonars and initializing the front wall and the right wall distances. This happens when the robot receives a command to move. The sonars cone is wider here because the sonars used here are different than the sonars used with 1D EKF. The sonars used in the 1D EKF were sonar1, sonar2, sonar11 and sonar12. In 2D EKF, the sonars used are sonar1, sonar3, sonar10 and sonar12. The reason is because in 2D EKF the robot is detecting walls or objects in different directions which is not only detecting the objects that are in the front of the robot as assumed in 1D EKF.

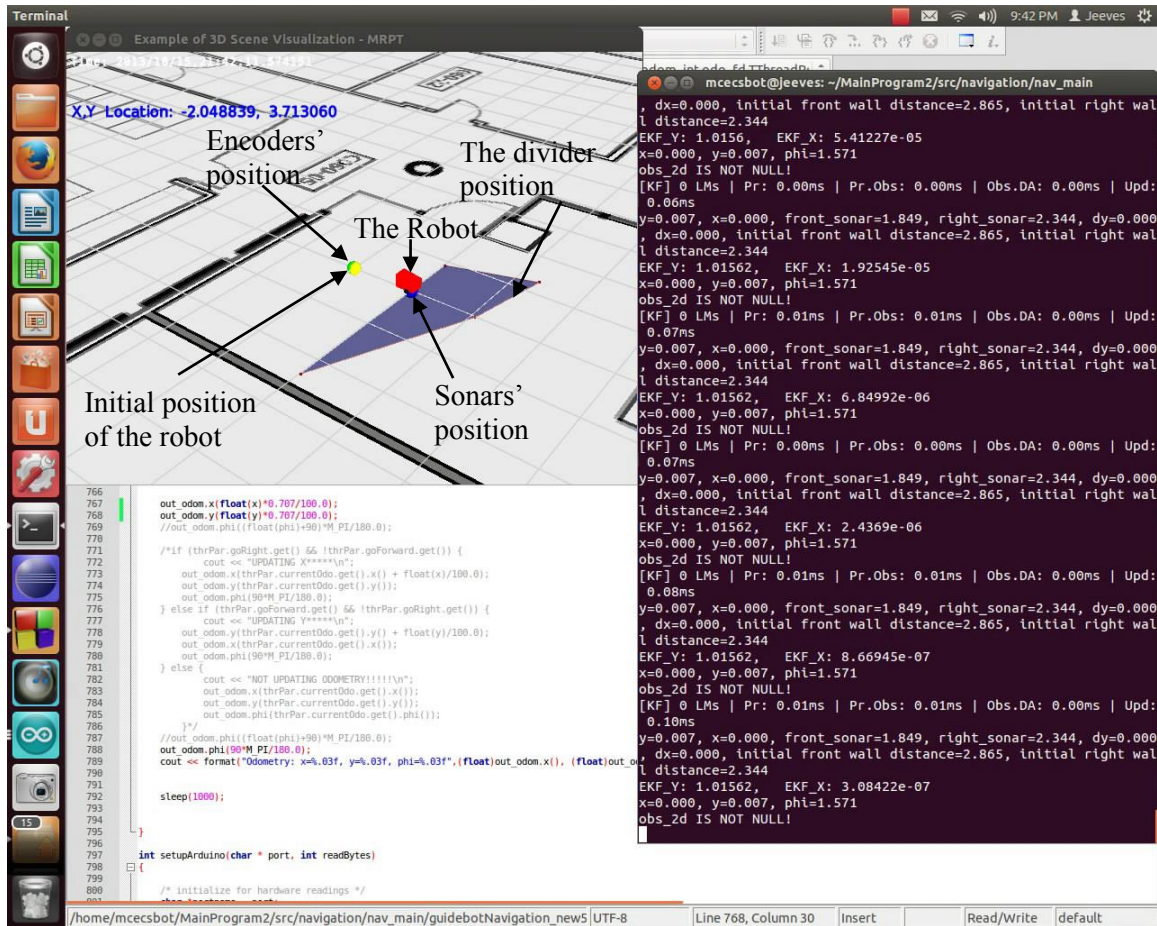


Figure 7.38: The first step of the robot is shown in the current figure. This figure shows that the position based on the sonars was updated but the one based on the encoders was not updated yet. However, the robot followed the sonars.

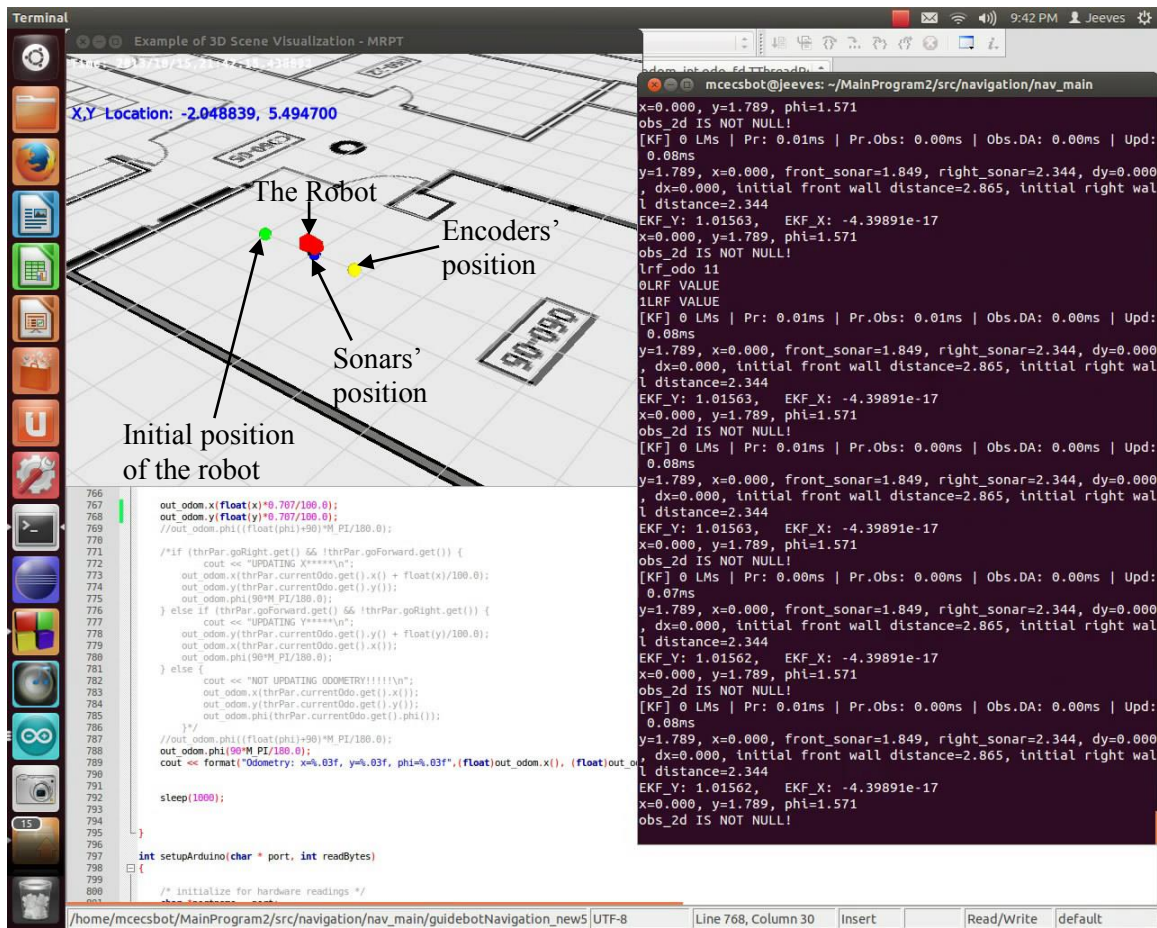


Figure 7.39: New object is shown here, which is the position of the robot based on the encoders' measurements.



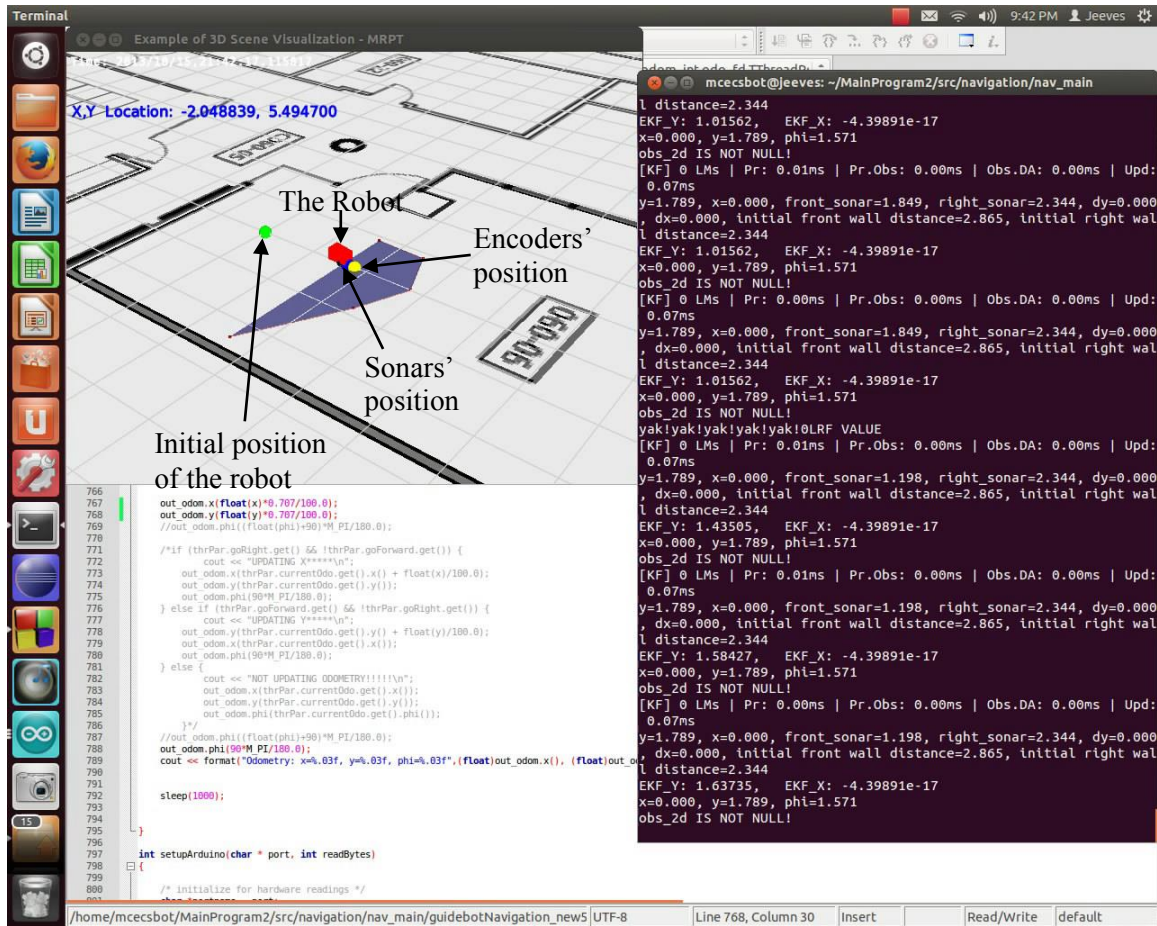


Figure 7.40: The robot's position based on sonars' measurements is updated here. The robot's position is closer to that based on sonars rather than that based on encoders. This is happening because the robot is affected by measurements (sonars) more than by the old estimate. This can be modified if the results is not satisfactory by changing the value of matrices  $R$  and/or  $Q$ .

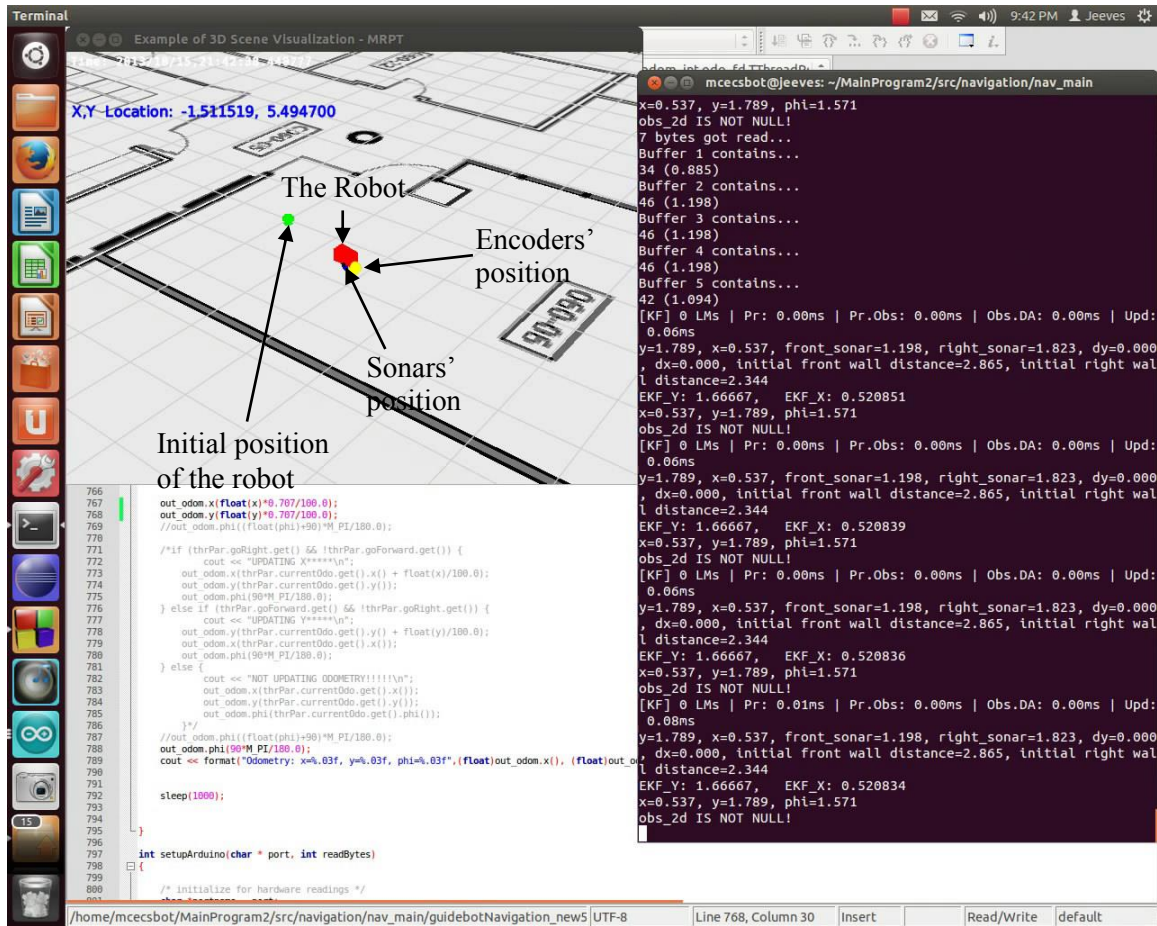


Figure 7.41: The robot start to move right here. Both position based on encoders' and sonars' were updated here.



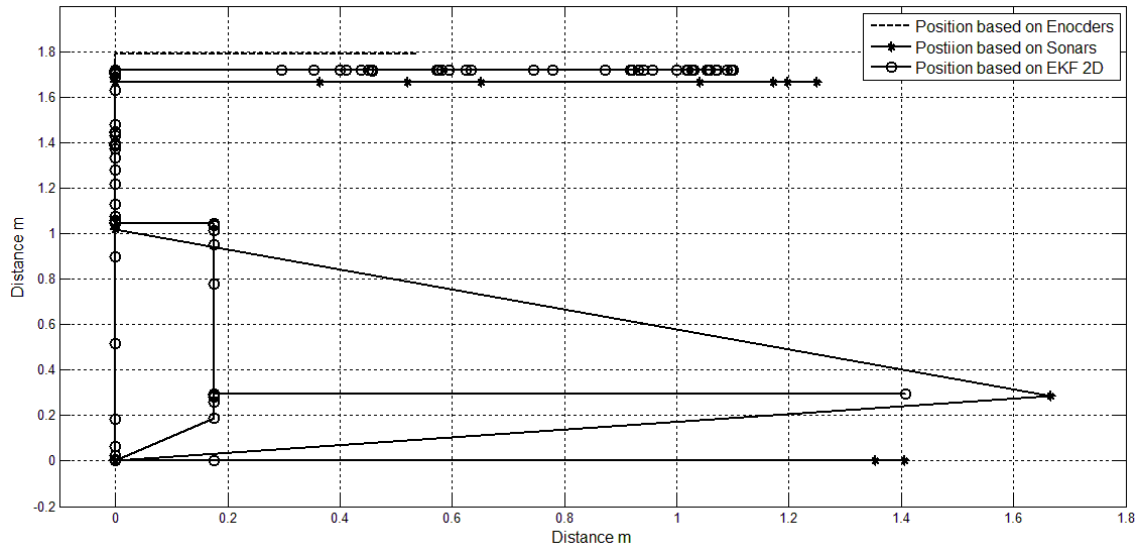


Figure 7.43: As shown, the estimate was little bit affected by the wrong measurements of the sonars. However, it wasn't affected for long time. The final estimation of the robot's position on x-axis is smaller than the one based on right sonar's measurements.



Chapter 8: Conclusion, Future Work and My Contribution

## **Chapter 8: Conclusion, Future Work and My Contribution**

This is the last chapter in this thesis. A conclusion to the previous chapters, plans for the future work and what my contribution to this project was are listed in this chapter.

### **8.1 Conclusion**

Kalman Filter is one of the most important algorithms that have been used with robots' localization. It has survived for more than half decade and is still applicable for robotic applications. Figure 8.1 shows a quick review of the structure and the content presented in this thesis. This thesis covered different topics that related to Kalman Filter. The diversity of examples give the readers a clear understanding to these different algorithms and concepts. Complicated subjects become easier to digest with easier and practical examples. Moreover, these examples are related to our robot. When the subjects become more advanced and closer to Kalman filter, the examples were about MCECS-bot and made from real data collected during robot operation.

Matlab was used to simulate these data beside the code that was used to test the robot (I mean MRPT code). The reason of giving such simulations is because Matlab code is easier and shorter than C++ code. Different scenarios were simulated with Matlab. These scenarios were presented with different filters before introducing Kalman Filter to them. This will clarify and show the importance of Kalman Filter. In this thesis, it has been proved that Kalman filter is a good algorithm to be used with linear and nonlinear systems. Kalman Filter was a good “refiner” when it comes to refine the information collected form sensors from noise. That is, because the gain of Kalman Filter is changing every recursion (it is not

fixed as in other filters like LPF). Additionally, Kalman Filter was able to fuse the sensors information and extract their strength. This is similar to combine sensors to a one more accurate sensor. This feature was successfully tested in Matlab and applied on MCECS-Bot. The code and hopefully my thesis will be available to other student in order to give them experience with MCECS-Bot and allows to improve upon my work.

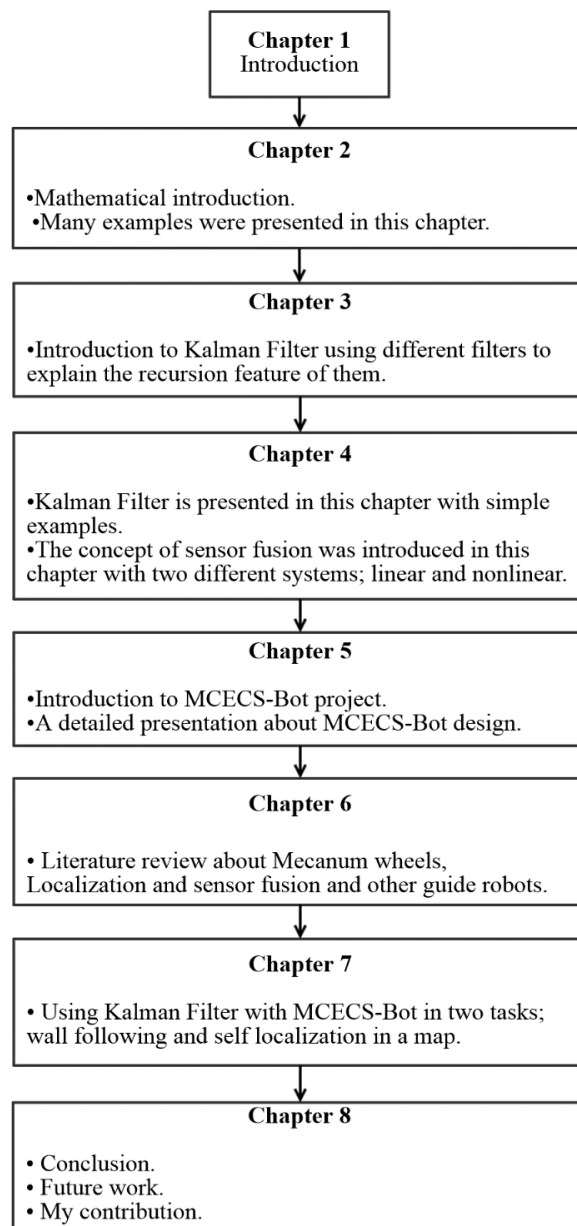


Figure 8.1: The structure of Mobile Robot Localization based on Kalman Filter thesis.

When a comparison is done between MCECS-Bot and other guide robots, it is obvious that the system of MCECS-Bot is more advanced. MCECS-Bot is equipped with better sensors when it comes to quantity and quality. MCECS-Bot has Mecanum wheels which is the main advantage and the best feature of its base. The robot has more freedom of base motion and body behavior than traditional guide robots. It has more means to solve the localization problems that will be encountered by MCECS-Bot during its operation. This project will attract students to work on navigation and other guide-bots functionalities in the future.

Whatever listed in this thesis will be a good resource for future student. I just have opened the door of navigation-related research. Therefore, my colleges don't have to start it from zero. The MRPT code is a good program that can be used and improved for MCECS-Bot. It is not that much complicated as we thought before. I believe that MRPT will be easier to work and experience with for other future students. The 1D EKF and/or 2D EKF are pieces needed to be put together in order to have a complete localization and navigation system for the MCECS-Bot. In the future work section, I will state how to implement a complete localization system based on EKF which takes its information from the map and the sensors.

One of the main goals of having this robot is education about guide-bot type of robots. Actually, educational robots are very simple like a robot arm or a sonar-based mobile robot. Guide-bots are sophisticated robots build by commercial companies or top universities and their internal contents is not presented to external audiences and definitely not treated as an educational material. In this sense MCEC-Bot and this thesis are a unique

concept that can find no counter-part in the world – create a sophisticated guidebot that serves also educational mission and that hardware and software can be completely reproduced by other universities and even high schools in future. All documentation and software will be available to others outside PSU with the aid of internet. In addition, our robot can be also controlled remotely by external users located all over the world. They will be able to use Kinect and cameras to see what the robot actually sees in the building. Does this experience affect the students' interest about the robots? Is it good for our robot to be surrounded by other students other than the students of PSU? Is it good for the engineering department to be an interesting place because of such projects? I believe the answers for these questions will be “yes”.

## **8.2 Future Work**

There are different areas in MCECS-Bot need to be improved and updated in the future. I will not write about the people interaction, the robot physical look or the upper part mechanical parts as they are topics of research of another student. Here, I will state whatever related to robot's localization, which includes whatever pieces that combine the robot base. The areas that need to be improved are:

### **8.2.1 Sonars**

The sonars have the problem of getting zero measurements as explained in chapter five. The solution now is using KF as a refiner to remove the noise from the coming signals. This method seems to be good but it would be better if there exist a hardware solution to this issue. First and foremost, let's specify what the noise source is in our system. Because

there is no noise when the robot is stationary and the noise appeared when the robot started to move, the sources of noise are motor controllers, cables that supply power to motors and the motors themselves. Accordingly, I recommend the following:

- 1- Replace the data cables of the sonars with a shielded wires. This will prevent or reduce the signal interfering from the motor controllers and the power cables as well as the noise coming from the motors themselves. Otherwise, a shield must surround the data cables coming from sonars.
- 2- The Arduino boards should be moved away from the noise source (the three main items I mentioned above).
- 3- The new shield should be connected to the battery's ground.
- 4- Sonars' orientation must point in parallel with the ground. Therefore, the angle between sonars' orientation and the ground level is not bigger than 5 degree. This will prevent having echoes reflected from the ground.

### **8.2.2 Encoders**

Currently, only two encoders are being used. For exploiting the advantages of Mecanum wheels, all encoders should be used. The figure (Figure 8.2) below reminds us of the features of Mecanum wheels. From this picture, multiple profiles can be extracted for encoders.

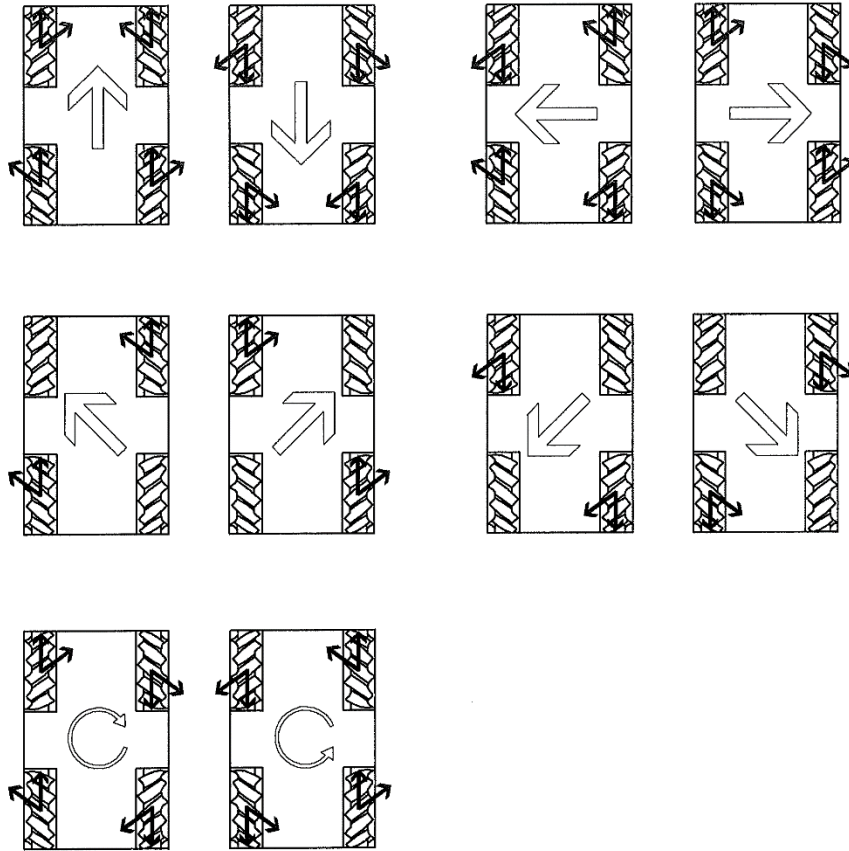


Figure 8.2: Different movement profiles of Mecanum wheels [23].

### 8.2.3 Battery and Velocity Indicator

The battery needs an indicators which can be LEDs that indicate the voltage level in the battery. Moreover, an input can be taken from each cell's terminal. This input can be interpreted in Arduino and presented later on the main monitor. Speed can be presented through the main monitor or by using a separate screen that shows the speed all the time.

#### **8.2.4 Advanced Localization based on Extended Kalman Filter**

The localization methods presented in chapter seven were about using EKF inside a room or in a short corridor. The one presented here is a robot's localization in a building map. Simply, using EKF to estimate the robot's position in a room but the initial position was in a different room. The information coming from encoders will be used to estimate the global position. On the other side, the information coming from the sonars is used to estimate the local position. Combining the global position with the local position using EKD will solve the problem of robot's localization in a building map. Additional data can be accessed in the Samsung Galaxy tablet, which are accelerometer and compass data. Figure 8.3 shows a map of a building (arbitrary map) with applying the new method of robot's localization on it. Figure 8.4 shows the information accessed from the tablet (This was done by Rami Alshafi but it is not implemented on MCECS-BOT yet).



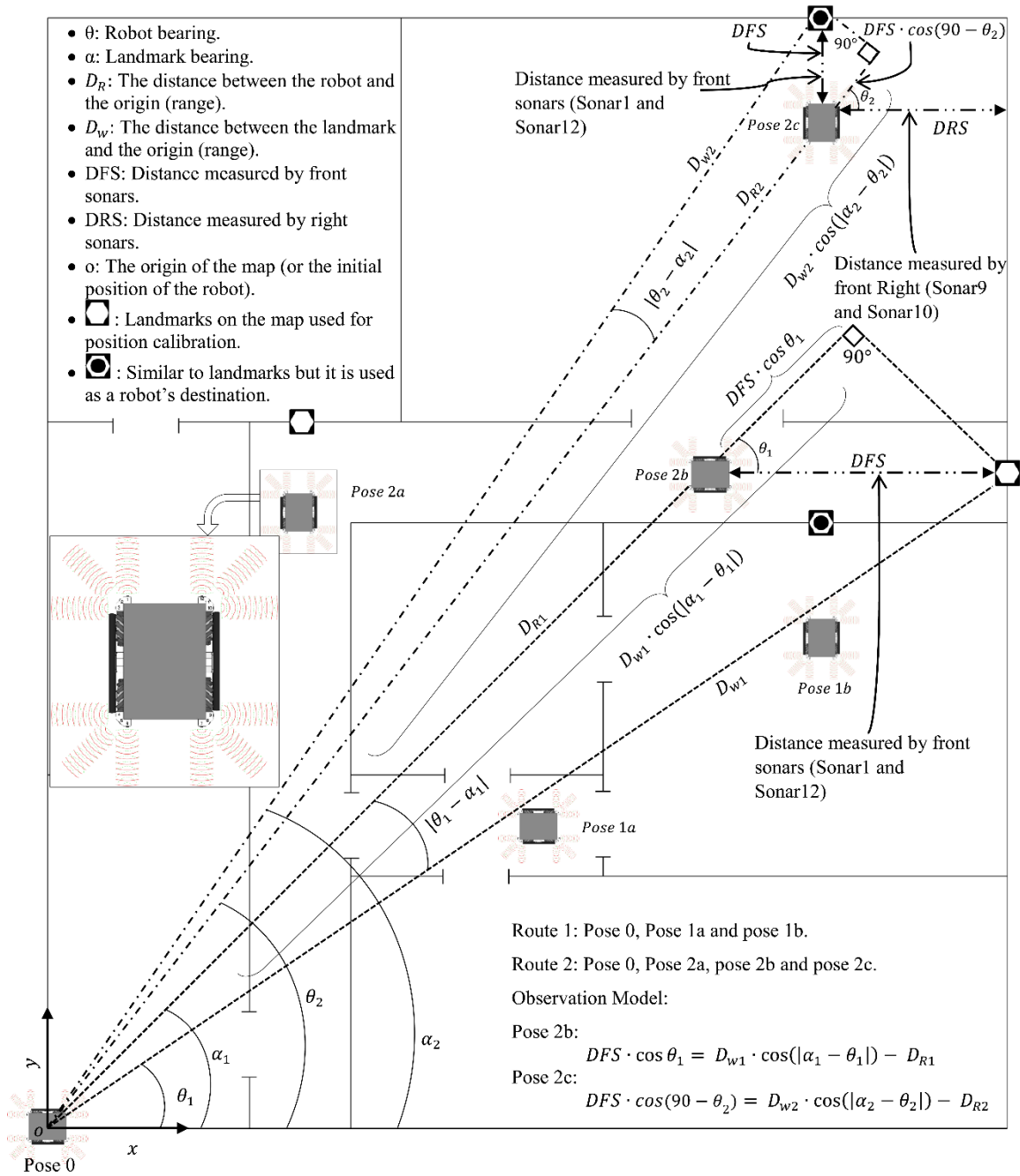


Figure 8.3: This figure shows a Map of a building and all the equations related to the robot's poses which are used later to derive equations' model of the EKF.

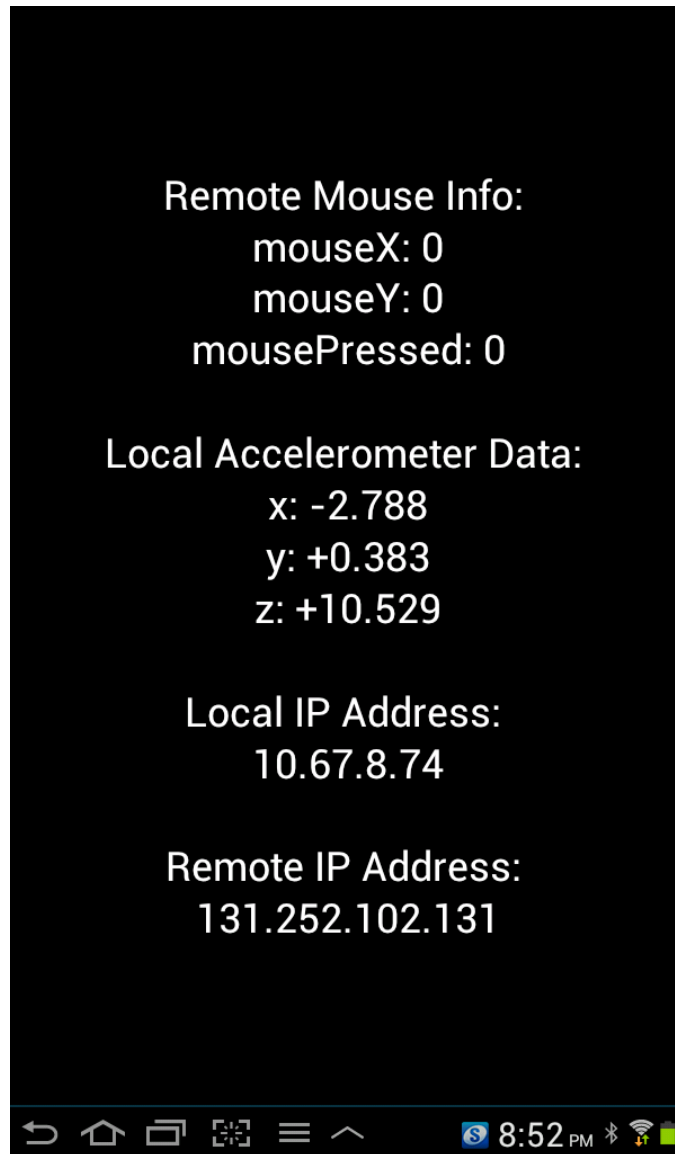


Figure 8.4: Data accessed on the Samsung Galaxy tablet.

As seen in Figure 8.3, there are two routes. The second route is used to illustrate the derivation of the system equations. Let's take pose 2b. The distance from the initial position and pose 2b is  $D_{R1}$ . A landmark is used to combine the local location (based on the distance from the surrounding walls) with the global localization (based on the distance

away from the initial position). The distance between this landmark and the initial position is  $D_{w1}$ . This distance is extracted from the map, which is provided by MRPT. The distance between the front wall and the robot is measured by sonars which is DFS. The relationship between sonar measurements and the global location that is provided by encoders is:

$$DFS \cdot \cos \theta_1 = D_{w1} \cdot \cos(|\alpha_1 - \theta_1|) - D_{R1} \quad \text{Equation 8.1}$$

The same thing goes for the third pose in the second route (pose 2c). The relationship between the local information of this pose and its global location is shown in equation 8.2:

$$DFS \cdot \cos(90 - \theta_2) = D_{w2} \cdot \cos(|\alpha_2 - \theta_2|) - D_{R2} \quad \text{Equation 8.2}$$

Equation 8.1 and Equation 8.2 will be used later to derive the observation model of the Extended Kalman Filter. One more input can be added to differentiate the robot's orientation. This information can come from a separate compass which is available from the tablet (Shown in Figure 8.4).

Finally, Wi-Fi can be used in this solution in order to give an additional constrain to increase the robot's confident about its position. The solution of using Wi-Fi is proposed by the current term Fall 2013 students of the Robotics I class (ECE578) in the Electrical and Engineering Department at PSU. When the robot gets closer to the router, it will sense a stronger signal, which can be used in the localization process (the signal strength is illustrated in Figure 8.5). I believe that using the

Figure 8.5: Wi-Fi signal strength representation.

Wi-Fi to improve the robot localization is a good idea and can be implemented with the proposed solution in this section<sup>15</sup>.

### **8.2.5 Connecting the MCECS-Bot to the Internet**

I have done a project in ECE 578 in Fall 2011, which was connecting a robot called Newton to the pave website designed by computer science faculty Dr. Fei Xie and his students. I wrote the code that does this task. Newton has four Omni wheels and other sensors. It was equipped with three cameras; the first one was placed in the front of the robot, the second one was placed on the back of it and the third one placed externally to show the whole robot and its surrounding environment. People from anywhere around the world can access the computer science webpage and find three screens of video streaming showing what the robot sees and the whole room that the robot is moving in. Additionally, there are command buttons to control the motion of the robot. The result is that people can control and see the robot's response. It would be not difficult to adapt this software to MCECS-Bot.

---

<sup>15</sup> In the below link, Localization using Wi-Fi Signal Strength is proposed, but they used Particle Filter instead of Kalman Filter. However, Extended Kalman Filter can be used to fuse Wi-Fi with the existing sensors. [http://robotics.usc.edu/~ahoward/projects\\_wifi.php](http://robotics.usc.edu/~ahoward/projects_wifi.php)

### **8.2.6 Testing for a Long Time Operation**

Add extensive testing for long period of time. This test will show the reliability and durability of the robot. For instance, we can claim that we have operated our robot for a thousand hours without maintenance.

### **8.3 My Contributions in this Thesis**

There are many students who have worked and continue working on MCECS-Bot. Therefore, it is important to list my contributions to this project. My contributions are:

- 1- I authored the entire design of the robot. The original conception of the robot by Dr. Perkowski was different and I convinced him to Mecanum wheels and the actual design.
- 2- I designed, modeled, simulated and build the robot's base. That includes the design of the base using aluminum bars and putting all the following components together. The main components of the robot's base are Mecanum wheels, Encoders and their gears, Motor-controllers, Motors, Gearboxes, Battery, Power Management Circuits and sonars. Choosing the right wheels, motors and gearboxes was based on calculations that I did.
- 3- I have chosen a compatible charger for the battery.
- 4- I contributed to mounting the LRF on the base and to use the LRF's measurements to simulate the Kinect data.
- 5- The power distribution board was done by Phil Lamb. It has some problems which were described in the fifth chapter. I fixed these problem and added other

functions to the power distribution board. I added many LED's to indicate the inputs and outputs of the PDB.

6- I designed the delay circuit that used to implement the chaining method that I used in connecting the twelve sonars. This circuit was used to delay the power to the second group of six sonars. This problem was described in detail in the fifth chapter.

7- I designed the robot's bumpers and building them from scratch.

8- I contributed in designing and constructing the waist with David Gaskin.

9- I designed the upper part of the robot that holds the neck.

10- I created a 3D model for MCECS-Bot. the program used is Google sketchUp.

This model can be found on the google 3d warehouse website. Anyone can download it and look at it using the google sketchUp program, which is a free software provided by google.

11- I created the wall following flowchart with Mathias Sunardi help. Later, I wrote the code for wall following, also with Mathias help.

12- I wrote all the Matlab codes used in this thesis.

## References

- [1] M. Perkowski, M. Sunardi, J. Larson, R. F and M. Lowe, "https://projects.cecs.pdx.edu/projects/roboticsclub-mcecsbot/wiki/Navigation," PSU, Portland, 2011.
- [2] "Math is Fun," 2012. [Online]. Available: <http://www.mathsisfun.com/data/standard-deviation.html>. [Accessed 4 April 2013].
- [3] "Math is Fun," 2013. [Online]. Available: <http://www.mathsisfun.com/data/probability.html>. [Accessed 7 April 2013].
- [4] S. Khan, KHANACADEMY, 20 2 2011. [Online]. Available: [https://www.khanacademy.org/math/probability/independent-dependent-probability/old\\_prob\\_videos/v/probability--part-7](https://www.khanacademy.org/math/probability/independent-dependent-probability/old_prob_videos/v/probability--part-7). [Accessed 22 3 2013].
- [5] C. Shih and G. Kochanski, "Bayes' Theorem," 15 September 2006. [Online]. Available: <http://kochanski.org/gpk/teaching/0401Oxford/Bayes.pdf>. [Accessed 11 May 2013].
- [6] G. Welch and G. Bishob, An Introduction to the Kalman Filter, Chapel Hill: SIGGRAPH, 2001.
- [7] S. Khan, "Probability Density Functions," KhanAcademy, 20 February 2011. [Online]. Available: [https://www.khanacademy.org/math/probability/random-variables-topic/random\\_variables\\_prob\\_dist/v/probability-density-functions](https://www.khanacademy.org/math/probability/random-variables-topic/random_variables_prob_dist/v/probability-density-functions). [Accessed 10 March 2013].
- [8] "Probability Distribution," Wikipedia, July 2011. [Online]. Available: [https://en.wikipedia.org/wiki/Probability\\_distribution](https://en.wikipedia.org/wiki/Probability_distribution). [Accessed 21 December 2012].
- [9] S. Khan, "ck12.org Normal Distribution Problems: Qualitative sense of normal distributions," KhanAcademy, 20 February 2011. [Online]. Available: [https://www.khanacademy.org/math/probability/statistics-inferential/normal\\_distribution/v/ck12-org-normal-distribution-problems--qualitative-sense-of-normal-distributions](https://www.khanacademy.org/math/probability/statistics-inferential/normal_distribution/v/ck12-org-normal-distribution-problems--qualitative-sense-of-normal-distributions). [Accessed 20 March 2013].
- [10] "Types of Distributions," AllPsych , 29 11 2011. [Online]. Available: <http://allpsych.com/researchmethods/distributions.html>. [Accessed 12 10 2012].
- [11] L. Kleeman, "Understanding and Applying Kalman Filtering," Department of Electrical and Computer Systems Engineering, Clayton.
- [12] P. Kim, Kalman Filter for Beginners with Matlab Examples, A-JIN, 2010.
- [13] CHRobotics, "Understanding Euler Angles," [Online]. Available: <http://www.chrobotics.com/library/understanding-euler-angles>. [Accessed 15 June 2012].
- [14] M. Sunardi, "MCECS Guide Robot Project," PSU, Portland, 2012.

- [15] MCECS Bot team, "Wiki/MCECS Bot," 15 June 2013. [Online]. Available: <https://projects.cecs.pdx.edu/projects/roboticsclub-mcecsbot/wiki/Navigation>.
- [16] "P60 Gearbox: Stock, Standard Shaft, RS-540/550 Mount, 64:1," BaneBot, 2012. [Online]. Available: <http://banebots.com/pc/P60K-S5/P60K-444-0004>. [Accessed 15 6 2012].
- [17] N. Tlale and M. de Villiers, "Kinematics and Dynamics Modelling of a Mecanum Wheeled Mobile Platform," in *15th International Conference on Mechatronics and Machine Vision in Practice (M2VIP08)*, Auckland, New-Zealand, 2008.
- [18] O. Mohsin, "PSU Guide Robot," 16 March 2012. [Online]. Available: <http://sketchup.google.com/3dwarehouse/details?mid=2d5e490d368bede3ed1231cfd3b598dd>. [Accessed 15 June 2013].
- [19] F. Adăscăliștei and I. Doroftei, "Practical Applications for Mobile Robots based on Mecanum Wheels - a Systematic Survey," *MECAHITECH'11*, vol. 3, pp. 61-63, 2011.
- [20] "Quad-Wheel Omni-Directional Vectoring Robot Kit," SUPERDROID ROBOTS INC, 2012. [Online]. Available: <http://www.superdroidrobots.com/shop/item.aspx/quad-wheel-omni-directional-vectoring-robot-kit/782/>. [Accessed 25 12 2012].
- [21] "4WD Arduino Compatible Mecanum Robot Kit," Nexus robot, 2010. [Online]. Available: [http://www.nexusrobot.com/product.php?id\\_product=67](http://www.nexusrobot.com/product.php?id_product=67). [Accessed 14 8 2012].
- [22] M. West and H. Asada, "Design of Ball Wheel Mechanisms for Omnidirectional Vehicles With Full Mobility and Invariant Kinematics," *Journal of Mechanical Design*, no. 119, pp. 153-161, 1997.
- [23] C. H. Kim, H. . T. Yang, S. K. Ha, H. C. Kim, D. Y. You, W. Lee and S. H. Han, "Width variable structure of moving and transport means using mecanum wheels". United States Patent US20130068543 A1, 21 03 2013.
- [24] "RoboClaw 2x15A," Pololu Robotics and Electronics, 2012. [Online]. Available: <http://www.pololu.com/catalog/product/1496>. [Accessed 20 June 2012].
- [25] "RS555 Motor - 12V," BaneBots, 2012. [Online]. Available: <http://banebots.com/pc/MOTOR-BRUSH/M5-RS555-12>. [Accessed 25 1 2012].
- [26] "COM-10932," Sparkfun Electronics, [Online]. Available: <https://www.sparkfun.com/products/10932>. [Accessed 1 June 2012].
- [27] "GBS-LFMP60AH (4 cell pack)," RebirthAuto , [Online]. Available: <http://rebirthauto.com/shop/batteries/gbs-lfmp60ah/>. [Accessed 29 July 2013].
- [28] "LFMP60AH," Electric Motor Sport, 2010. [Online]. Available: [http://www.electricmotorsport.com/store/ems\\_ev\\_parts\\_batteries\\_lpf\\_gbs\\_60ah.php](http://www.electricmotorsport.com/store/ems_ev_parts_batteries_lpf_gbs_60ah.php). [Accessed 20 May 2012].
- [29] "MB1000 LV-MaxSonar," MaxBotix, 2012. [Online]. Available: [http://www.maxbotix.com/Ultrasonic\\_Sensors/MB1000.htm](http://www.maxbotix.com/Ultrasonic_Sensors/MB1000.htm). [Accessed 25 May 2012].



- [30] M. Barton, J. Barrett, C. O'Connell, J. Adams and P. Lamb, "MCECS Bot," 2012.
- [31] T. Bonar, "Using Multiple MaxSonar Sensors," MaxBotix, 07 11 2012 . [Online]. Available: <http://www.maxbotix.com/articles/031.htm>. [Accessed 15 08 2013].
- [32] A. Broughton, "AAARGH! Sonar Noise!," Big Georgia Spots, 20 February 2012. [Online]. Available: <http://www.biggeorgiaspots.com/boats-and-electronics/aaargh-sonar-noise.html?showall=1>. [Accessed 5 August 2013].
- [33] "Laser Range Finder," PARALLAX INC, [Online]. Available: <http://www.parallax.com/product/28044>. [Accessed 15 1 2013].
- [34] "Parallax Laser Range Finder (#28044) - Parallax, Inc.," 16 9 2011. [Online]. Available: <http://www.yumpu.com/en/document/view/11782645/parallax-laser-range-finder-28044-parallax-inc>. [Accessed 5 8 2013].
- [35] "Protection Circuit Module (PCM) for 4 cells (12.8V) LiFePO4 Battery Pack at 16A limited -- PCM-B04S30-190(LFP)," AA Power Portable Corp, [Online]. Available: <http://www.batteryspace.com/pcmprotectioncircuitmodulefor4cells128vlifepo4battery-pack-at-16a-limited.aspx>. [Accessed 22 Jun 2012].
- [36] "Protection Circuit Module (PCM) for 4 cells (12.8V) LiFePO4 Battery Pack at 16A limited -- PCM-B04S30-190(LFP)," AA Portable Power Corp, [Online]. Available: <http://www.batteryspace.com/pcmprotectioncircuitmodulefor4cells128vlifepo4battery-pack-at-16a-limited.aspx>. [Accessed 8 8 2012].
- [37] "Tenergy 12.8V (4S) 10A LiFePO4 Battery Charger," All Battery, [Online]. Available: <http://www.all-battery.com/Tenergy14.6V10ALiFePO4BatteryCharger-01034.aspx>. [Accessed 24 Jun 2012].
- [38] "List of MRPT libraries," MRPT, 2013. [Online]. Available: <http://www.mrpt.org/Libraries>. [Accessed 14 09 2013].
- [39] J. L. Blanco, "Kalman Filters," MRPT, 11 10 2013. [Online]. Available: [http://www.mrpt.org/Kalman\\_Filters](http://www.mrpt.org/Kalman_Filters). [Accessed 18 10 2013].
- [40] J. Kim, J. Park and S. Kim, "Inertial Navigation System for Omni-directional AVG with Mecanum Wheel," *Advances in Mechanical Engineering*, vol. 2, 2012.
- [41] S. I. Roumeliotis and G. A. Bekey, "Bayesian estimation and Kalman filtering: A unified framework for Mobile Robot Localization," in *International Conference on Robotics and Automation*, San Francisco, 2000.
- [42] S. Jia, A. Yasuda, D. Chugo and K. Takase, "LRF-Based Self-Localization of Mobile Robot Using Extended Kalman Filter," in *SICE Annual Conference*, Tokyo, 2008 .
- [43] F. Kong, Y. Chen, J. Xie, G. Zhang and Z. Zhou, "Mobile Robot Localization Based on Extended Kalman Filter," in *6th World Congress on Intelligent Control and Automation*, Dalian, China, 2006.
- [44] J. Kim, Y. Kim and S. Kim, "An Accurate Localization for Mobile Robot Using Extended Kalman Filter and Sensor Fusion," in *International Joint conference on Neural Network (IJCNN 2008)*, 2008.

- [45] J. Z. Sasiadek and P. Hartana , "Sensor Data Fusion Usin Kalman Filter," in *The Third International Conference on Information Fusion*, Paris, 2000.
- [46] Q. Gan and C. J. Harris, "Comparison of Two Measurements Fusion Methods for Kalman-Filter-Based Multisensor Data Fusioin," *IEEE Transaction on Aerospace and Electronic Systems*, vol. 37, no. 1, pp. 273-279, 2001.
- [47] M. Pinto, A. Matos and A. P. Moreira, "Localization of Mobile Robot Using an Extended Kalman Filter in a LEGO NXT," *IEEE Transactions on Education*, vol. 55, no. 1, p. 135–144, 2012.
- [48] W. Burgard, A. B. Cremers, D. Fox, D. Hahnel, G. Lakemeyer, D. Schulz, W. Steiner and S. Thrun, "The InteractiveMuseum Tour-Guide Robot," in *The Fifteenth National Conference on Artificial Intelligence*, Madison, 1998.
- [49] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte and D. Schulz, "MINERVA: A Second-Generation Museum Tour-Guide Robot," in *International Conference on Robotics & Automation*, Detroit, 1999.
- [50] N. Tomatis, R. Philippsen, B. Jensen, K. O. Arras, G. Terrien, R. Piguët and R. Siegwart, "Building a Fully Autonomous Tour Guide Robot: Where Academic Research Meets Industry," in *33rd International Symposium on Robotics*, Sockholm, Sweden, 2002.
- [51] M. A. Salichs, R. Barber, A. M. Khamis, M. Malfaz, J. F. Gorostiza, R. Pacheco, R. Rivas, A. Corrales, E. Delgado and D. Garcia, "Maggie: A Robotic Platform for Human-Robot Social Interaction," in *Automation and Mechatronics, IEEE Conference on Robotics*, Bangkok, 2006.
- [52] "RoboClaw 2x15A," Pololu Robotics & Electronics, [Online]. Available: <http://www.pololu.com/catalog/product/1496/resources>. [Accessed 15 4 2012].
- [53] "LFMP60AH," 2010. [Online]. Available: [http://www.electricmotorsport.com/store/ems\\_ev\\_parts\\_batteries\\_lpf\\_gbs\\_60ah.php](http://www.electricmotorsport.com/store/ems_ev_parts_batteries_lpf_gbs_60ah.php). [Accessed 11 4 2012].
- [54] "Material Safety Data Sheet," 22 1 2010. [Online]. Available: <http://www.electricmotorsport.com/store/pdf-downloads/GBS/MSDS-60AH-English.pdf>. [Accessed 17 8 2013].
- [55] "Tenergy 12.8V (4S) 10A LiFePO4 Battery Charger," All-Battery.com, [Online]. Available: <http://www.all-battery.com/Tenergy14.6V10ALiFePO4BatteryCharger-01034.aspx>. [Accessed 12 6 2012].
- [56] M. J. Baker, "Maths - Quaternions," Euclideanspace, [Online]. Available: <http://www.euclideanspace.com/math/algebra/realNormedAlgebra/quaternions/>. [Accessed 21 10 2013].

## Appendices

## **Appendix A: Practicle Applications for Mobile Robots based on Mecanum Wheels- a Systematic Survey [19]**

*Proceedings of International Conference On Innovations, Recent Trends And Challenges In Mechatronics, Mechanical Engineering And New High-Tech Products Development – MECAHITECH'11, vol. 3, year: 2011*

---

### **Practical Applications for Mobile Robots based on Mecanum Wheels - a Systematic Survey**

Florentina Adăscăliței, Ioan Doroftei

”Gh. Asachi” Technical University of Iasi, Mechanical Engineering Faculty, Theory of Mechanisms and Robotics Department, B-dul D. Mangeron, 61-63, 700050, Iasi, Romania  
E-mail: [adascalitei\\_florentina@yahoo.com](mailto:adascalitei_florentina@yahoo.com), [ioan\\_doroftei@yahoo.com](mailto:ioan_doroftei@yahoo.com)

#### **Abstract**

In this paper a literature review concerning practical applications for mobile robotic platforms based on special wheels (in this case, Mecanum wheel) is presented. Mobile robots equipped with four Mecanum wheels have the omnidirectional property, which means, they have the ability to move instantaneously in any direction, from any configuration. Therefore, compared to conventional platforms, these vehicles possess multiple advantages in terms of their mobility in narrow spaces or crowded environments. They have the ability to easily perform certain tasks in congested environments foreseen with static obstacles, dynamic obstacles or narrow areas. Usually, such environments are found in factory workshops, warehouses, hospitals, etc. Hence the resulting needs to create this kind of robotic platforms to satisfy the requirements of various fields, such as: industrial, military, naval, medical and last but not least, the educational field (as the basis for research). The characteristics of the Mecanum wheel, a short comparison between this type of wheel and a conventional wheel, as well as the constructive and design solutions previously developed are described in the first part of this paper. Then, some application fields and the related systems based on Mecanum wheel are presented.

#### **Keywords**

Mecanum wheel, omnidirectional mobile robot, AGV

#### **Introduction**

Omnidirectional wheels have been used in robotics, in industry, and in logistics for many years. By reviewing and analyzing systematically the existing literature concerning this type of wheels, it was revealed that systems based on Mecanum wheels detain omnidirectional capabilities, whereas systems based on conventional wheels do not. Specifically, these capabilities make the vehicle extremely maneuverable, which could be very helpful in different indoor and outdoor applications. Therefore, compared to conventional vehicles, omnidirectional robotic vehicles possess multiple advantages in terms of their mobility in narrow spaces and crowded environments. They have the ability to easily perform certain tasks in congested environments foreseen with static obstacles, dynamic obstacles or narrow areas. Usually, such environments are found in factory workshops, warehouses, hospitals, etc. Hence the resulting needs to create this kind of robotic platforms to satisfy the requirements of various fields, such as: industrial, military, naval, medical and last but not least, the educational field. Furthermore, to prevent the shortcomings presented by Mecanum

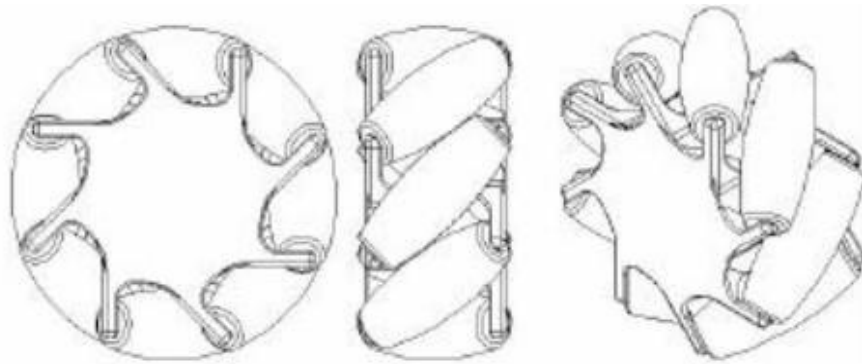
wheel, researchers have focused on its optimization, developing new constructive solutions, thus allowing their implementation in new applications, such as planetary explorations, mine operations.

## **Mecanum wheel**

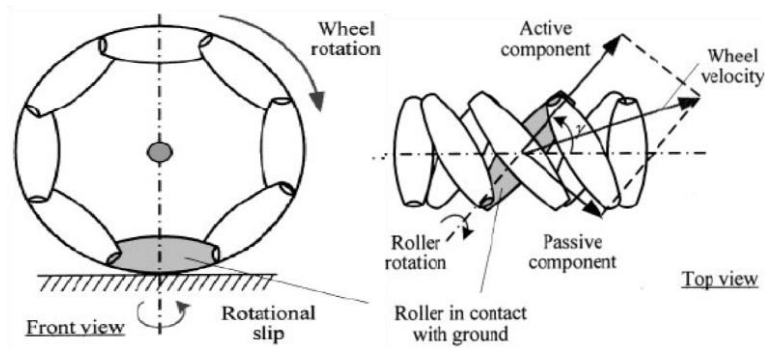
### **Mecanum wheel characteristics**

Mecanum wheel was designed and invented in Sweden, in 1975, by Bengt Ilon, an engineer with the Swedish company Mecanum AB [1]. Mecanum wheel is based on the principle of a central wheel with a number of rollers placed at an angle around the periphery of the wheel. The angle between rollers axis and central wheel axis could have any value, but in the case of conventional Mecanum wheel it is  $45^\circ$  (Figure 1). The rollers are shaped such that the silhouette of the omnidirectional wheel is circular. The angled peripheral rollers translate a portion of the force in the rotational direction of the wheel to a force normal to the wheel direction. Depending on each individual wheel direction and speed, the resulting combination of all these forces produces a total force vector in any desired direction, thus allowing the platform to move freely in direction of the resulting force vector, without changing the direction of the wheel.

A Swedish omnidirectional wheel has 3 DOF's composed of wheel rotation, roller rotation and rotational slip about the vertical axis passing through the point of contact (Figure 2). In the omnidirectional wheel, the wheel velocity can be divided into the components in the active direction and in the passive direction. The active component is directed along the axis of the roller in contact with the ground, while the passive one is perpendicular to the roller axis [2]. When the wheel rotates, a force vector along the wheel and a force vector perpendicular to the wheel are created. By a simple control of each wheel rotation, the vehicle moving direction can be changed instantaneously.



A. 1: Mecanum wheel.

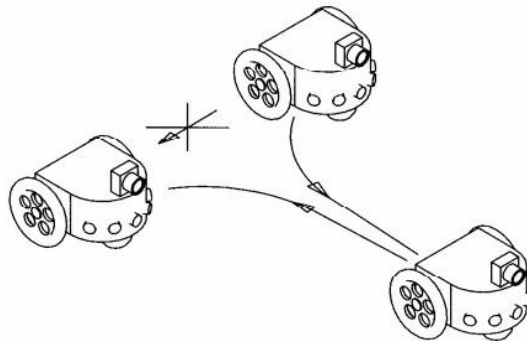


A. 2: DOF's in a Mecanum wheel [2].

When a Mecanum wheel is rotating, at least one roller (maximum two rollers) is (are) in contact with the ground. Only a small surface (theoretical, one point) of the roller is in contact with the ground. The area of this surface traverses the roller from one side to another, depending on the sense of wheel rotation. The direction of the traction force will be done by the traversing sense of contact surface. It means, if we look to the wheel from the top side, the traction force will be perpendicular to the roller axis [3].

### Mecanum wheel vehicle vs. Conventional wheel vehicle

The benefits of a vehicle with Mecanum wheels relative to one with steered wheels have been presented by [4]. Usually, robotic vehicles are designed to perform planar motion. In a two dimensional space, a body has three degrees of freedom, being capable of translating in both directions and rotating about its centre of gravity. However, most conventional vehicles do not have the ability to control every degree of freedom independently, because conventional wheels are not capable of moving in a direction parallel to their axis. These so called nonholonomic constraints of the wheel prevent vehicles using skid-steering from moving perpendicular to its drive direction. To reach every location and orientation in a two dimensional space it can require complicated maneuvers and complex path-planning. Non-holonomic vehicles can move in some directions (forward and backward) and can describe some curved trajectories, but cannot crab sideways. For example, to realize a parallel parking, a differential drive vehicle should make a number of maneuvers (Figure 3).

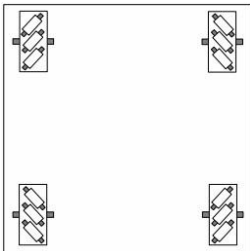
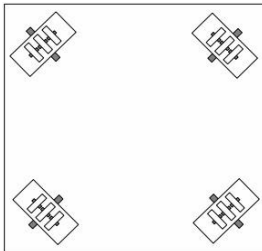
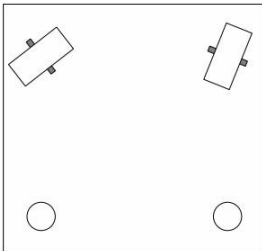


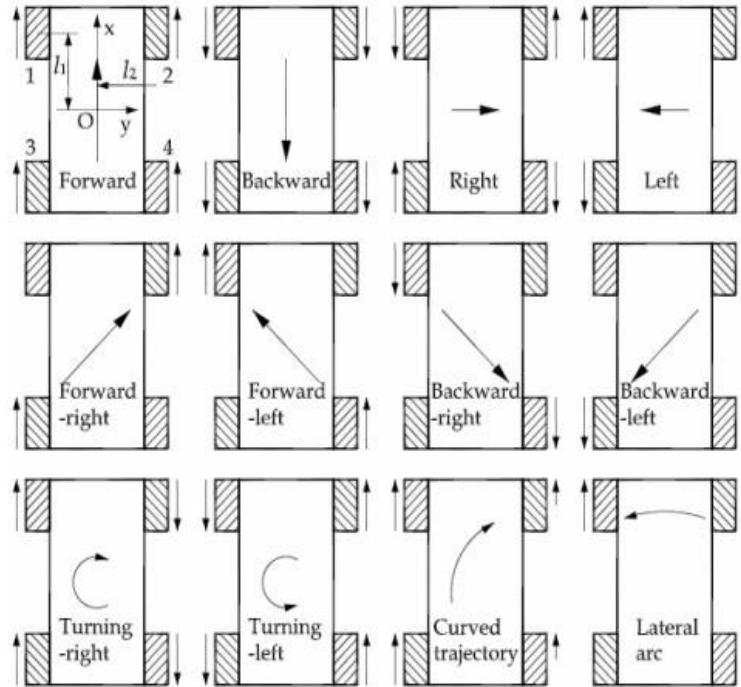
A. 3: Lateral parking of a differential drive mobile.

A vehicle without non-holonomic constraints it can travel in any direction under any orientation. This capability is widely known as omnidirectional mobility. Omnidirectional vehicles have great advantages over conventional platforms, with car-like Ackerman steering or differential drive system in terms of moving in tight areas [5]. They can crab sideways, turn on the spot and follow complex trajectories. These vehicles are capable of easily performing tasks in environments with static and dynamic obstacles and narrow spaces.

Usually, vehicles based on Mecanum wheel have a square or a rectangular configuration, with two wheels on each side of the chassis. Using four of these wheels provides omnidirectional movement for a vehicle without needing a conventional steering system. When Mecanum wheels are actuated, the angled peripheral rollers translate a portion of the force in the rotational direction of the wheel to a force normal to the wheel direction.

Table A. 1: Comparison between different types of drives.

	<b>Mecanum drive</b>	<b>Holonomic drive</b>	<b>Swerve drive</b>
<b>Description</b>	 <p>Wheels with angled rollers</p>	 <p>Wheels with “straight” rollers (omniwheels)</p>	 <p>Independently steered drive modules</p>
<b>Advantages</b>	compact design high load capacity simple to control less speed and pushing force when moving diagonally	low weight compact design simple to control less speed and pushing force when moving diagonally	simple conceptually simple wheels continuous wheel contact high load capacity robust to floor conditions
<b>Disadvantages</b>	very complex conceptually discontinuous wheel contact high sensitivity to floor irregularities complex wheel design	more complex conceptually discontinuous wheel contact or variable drive-radius sensitive to floor irregularities lower traction	complex mechanical design heavy and massive design complex to program and control high friction and scrubbing while steering



A. 4: Vehicle motion according to the direction and angular speed of the wheels [6].

Depending on each individual wheel direction and velocity, the resulting combination of all these forces produce a total force vector in any desired direction thus allowing the platform to move freely in the direction of the resulting force vector, without changing of the wheels themselves. The vehicle is able to translate on any direction, forward/backward but also sideways left/right and turning on the spot, thanks to its special wheels (Figure 4). This is especially helpful when having to maneuver in tight environments [5]. A short comparison between Mecanum drive, holonomic drive and swerve drive is presented in Table 1.

### Mecanum wheel constructive solutions

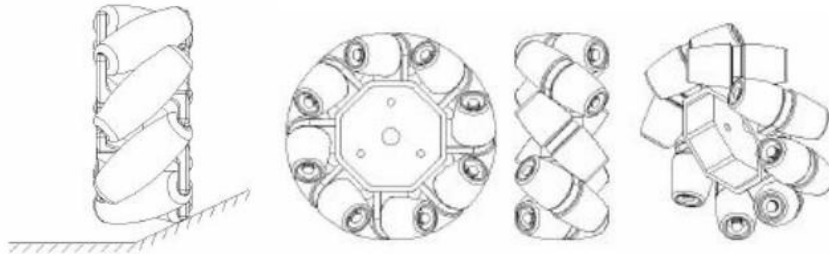
Omnidirectional wheeled vehicles with Mecanum wheels have some shortcomings. According to [7], a vehicle with Mecanum wheels is susceptible to slippage, and as a result, with the same amount of wheel rotation, lateral travelling distance is different from longitudinal travelling distance. In addition, the ratio of longitudinal travelling distance over lateral travelling distance with the same amount of wheel rotation, changes with ground condition. The second drawback is that the contact point between the wheel and the ground moves along a line parallel to the wheel axis, even though the wheel is always in contact with the ground. The lateral movement produces horizontal vibrations. The last drawback is that its ability to overcome obstacles is not independent of travel direction.

The slippage of the wheels prevents the most popular dead-reckoning method, using rotary shaft encoders [5], [8], from being performed well on a vehicle with Mecanum wheels. In order to solve the problem, visual dead-reckoning was used as a slip-resilient sensor [7], [9]. This technique, also used in optical mice, makes use of an on-board video-camera continuously capturing frames of the ground beneath and image processing hardware on the robot determining the speed and direction in which the current frame has moved relative to the previous frame thus allowing the speed and direction of that point of reference to be calculated.

A traditional Mecanum wheel with the peripheral rollers held in place from the outside is presented in Figure 1. This design, although having a good load carrying capacity, has the disadvantage that, when encountering an inclined or uneven surface, the rim of the wheel can make contact with the surface, instead of the roller,



therefore preventing the wheel from operating correctly (Figure 5.a). A simple alternative design, also proposed by Ilon, which alleviates the problem, consists in having the rollers split in two (or in three) and centrally mounted as shown in Figure 5.b. This design ensures that the rollers are always in contact with the work surface, thus allowing a better performance on uneven surfaces [10].

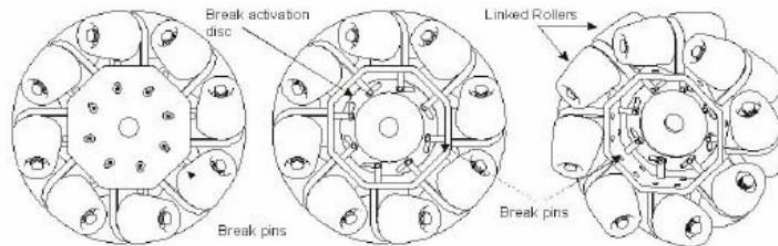


A. 5: a) Traditional Mecanum wheel on inclined surface; b) Mecanum wheel with centrally mounted rollers

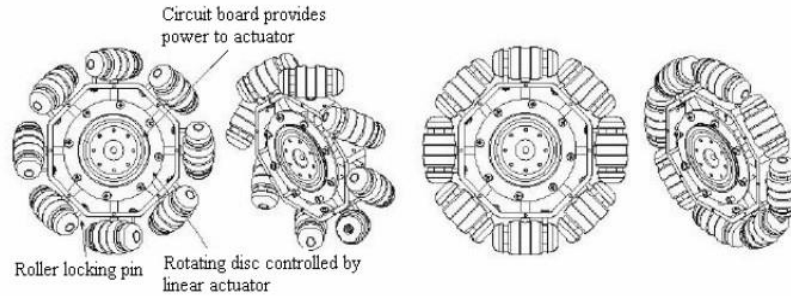
(a)

(b)

One disadvantage of the Mecanum design is the inefficient use of the kinetic energy supplied to the wheels by the motors. Due to the rotation of the exterior rollers, only a component of the force at the perimeter of the wheel is applied to the ground and the resulting force only partially contributes to the motion of the vehicle. [11] proposed two designs to improve the Mecanum wheel efficiency. The first design is the Mecanum wheel with lockable rollers illustrated in Figure 6. This design was conceived to overcome the losses of efficiency due to energy lost in a direction normal to that of travel through the peripheral rollers (they bleed off energy as they rotate), when the vehicle is travelling in a straight line (forward/backward). Simple actuators are used to rotate the brake activation disc, therefore to lock and unlock the roller, when the vehicle is moving. When driving in longitudinal motion, the peripheral rollers will be locked and they will act as a heavy thread, but when driving in sideways motion the rollers will be unlocked. This design is effective in reducing any lost forces in the forward direction to zero, but does not improve the losses in any other directions.



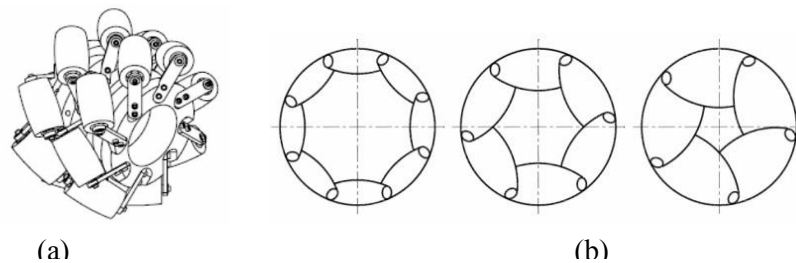
A. 6: Mecanum wheel with lockable rollers [11].



A. 7: Mecanum wheel with rotatable rollers [11].

The second design is Mecanum wheel with rotatable rollers illustrated in Figure 7. Compared to the first design, this one is more effective, but mechanically more complex. The peripheral rollers are split and centrally mounted on an axle which can be pivoted through  $135^\circ$ . This allows the rollers to be adjusted from a straight position (in which they are locked so the rollers cannot rotate on their axles), thus effectively forming an almost normal treaded tire, to an angle of  $45^\circ$  in which case they act as a traditional Mecanum wheel, or to an angle of  $135^\circ$ , making diagonal travel easier as it overcomes the resistance given by the traditionally immobile wheels. The angle of the rollers on each wheel is controlled through all the roller shafts, which are connected through a bevel gear system in such a way that a rotary actuator on one of the shafts controls all the other simultaneously.

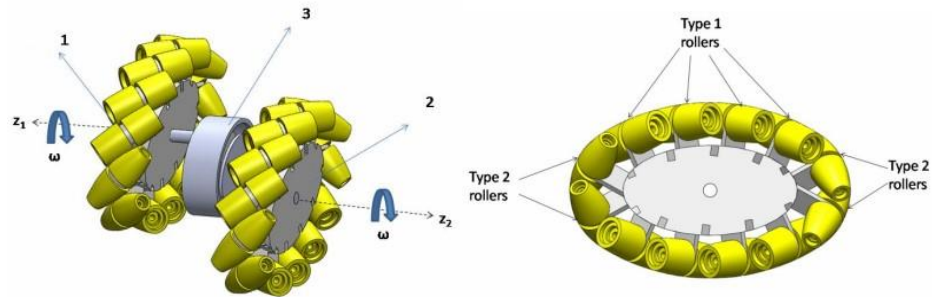
[3] proposed a new Mecanum wheel constructive solution in terms of its performance on various surfaces and concluded that the size of the peripheral rollers has a great effect upon this performance (Figure 8.a). The larger the rollers are, the greater the range of surface deviations can be overcome. Also, as the size of rollers increases, the slower they spin, resulting in lower friction losses in the driving of the wheel. In conclusion, when designing a new drive system for a vehicle, there exist a certain number of rollers that makes the ideal compromise between having a small number of large rollers per wheel, and having a large number of small rollers per wheel (Figure 8.b).



A. 8: a) New constructive wheel design; b) Rollers number according to their size.

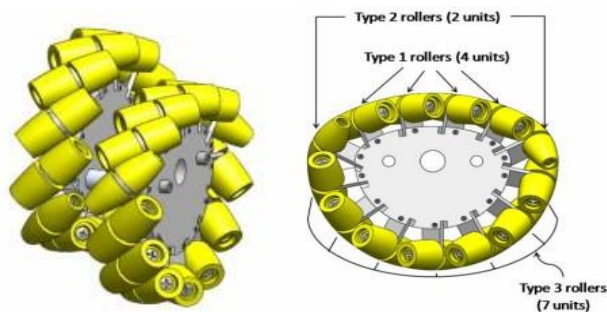
Large size of rollers means a small number of them. This has as effect a very small radius to the rollers extremities and, in this case, it could be difficult to use ball bearings in order to decrease the friction between the roller and its axis. The new constructive solution not only facilitates the use of big bearings, but also makes possible the approximation of the roller shape with a circle, because the roller length becomes smaller than the one used in a traditional Mecanum wheel – it is half of the normal roller.

In order to overcome the Mecanum wheel difficulties when moving on rough terrain, [12] has developed two new concepts of this type of wheel, using the principle of “steeping on obstacles”. The first one is the concept of “elliptical Mecanum double wheel” (Figure 9). The wheel itself consists in two elliptically Mecanum wheels, 1 and 2, coupled with a special mechanism in between 3. The role of the mechanism 3 is to move (rotation and relative translation to each other) the components 1 and 2 in such a way that the resulting motion would provide the same movement (in terms of speed, direction, smoothness) to the vehicle as a traditional Mecanum wheel would. For practical purposes, components 1 and 2 are not truly elliptical, when viewed sideways, but rather an approximation of an ellipse is used. The designed wheel includes a total number of 12 rollers (8 of type 1 and 4 of type 2). The maximum obstacle height that this wheel can overcome is 75% of the wheel largest radius.



A. 9: Assembled elliptical Mecanum double wheel [12].

The second concept is the “semicircular Mecanum double wheel”. The idea was to modify the elliptically shaped wheel seen in Figure 9, and replace it with a wheel having half circular and half elliptical profiles as shown in Figure 10. In this case, as the double wheel rotates, the smooth motion of a regular Mecanum wheel is achieved, while the ability of overcoming small obstacles when travelling laterally is kept. In contrast to the elliptical wheel, the semicircular wheel includes three types of rollers (4 of type 1, 2 of type 2 and 7 of type 3) and the total number of rollers is 13. The flatter the elliptical part is the greater clearance can be achieved for overcoming higher obstacles. This type of wheel is capable of overcoming obstacles of height up to 37.5% of the wheel’s radius. Also, both of these types of wheel have the ability of moving in soft dirt. The wheel would pile up a small amount of dirt and then step on it, thus avoiding being blocked by large amounts of dirt.



A. 10: Assembled semicircular Mecanum double wheel [12].

## Practical applications in various fields

### Military field

The manoeuvrability provided by omnidirectional vehicles can be utilized and can be very important in numerous outdoors applications, such as search and rescue missions, military activities, planetary explorations and mine operations.

This wheel is commonly used in robotic applications requiring a high degree of maneuverability, such as those experienced by NASA for hazardous environment exploration [13]. The objective of the OmniBot project (Figure 11) is to develop a hazardous duty mobile base as an advanced development test bed to research alternate technical approaches for remotely controlled operations in hazardous areas. In addition, this base will be used to test various automated umbilical technologies for autonomous mobile vehicles.



A. 12: NASA OmniBot mobile base [14].

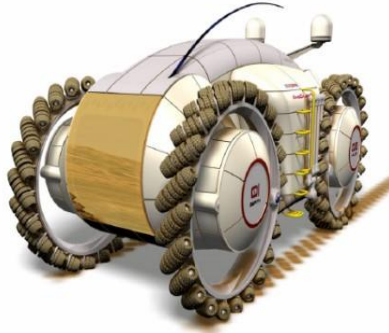


A. 11: Mecanum wheel vehicle for USA Navy [15].

In hazardous environments where it is too dangerous to send in unprotected personnel, a mobile base could be used to perform remote inspections, site surveys, and operations. The OmniBot is driven with four brushless servomotors connected to omnidirectional wheels (Mecanum). This allows for complete 2-degree-of-freedom motion, which results in extremely high maneuverability. The benefit of this motion profile can truly be appreciated when the vehicle is operated in a teleoperational mode. The vehicle can be controlled with a radio frequency (RF) control box or with a hardwired joystick. With the video transmission gear installed, teleoperation is possible up to a distance of 1,800 feet [14].

*Omnix Technology Systems, Inc.* had developed Mecanum wheel vehicle for U.S Navy for inspection of areas inaccessible to humans and vehicles capable of transporting very heavy loads in military environments [15]. These vehicles can be seen in Figure 12 are especially adaptable for autonomous or teleported operations due to the unrestricted manoeuvrability and simplicity of control.

*MarsCruiserOne* is a pressurized, habitable rover, designed to allow exploration of the Moon and Mars during future space missions (Figure 13). Characterized by omnidirectional wheels especially suited to tackle rocky terrain, it travels at a speed of 5-10 km/h [16]. This design incorporates: hubless wheels (which allowed ingress/egress for the astronaut crew for extra-vehicular activities and access to other surface modules and rovers), Mecanum wheels, a linear motor drive and a single point rotary shock absorber/suspension system [17].



A. 13: MarsCruiserOne [16].

### **Industrial field**

Airtrax ATX-3000 industrial forklifts (Figure 14) excel in applications requiring tight manoeuvring or transporting long loads sideways through standard sized doors or narrow aisle ways. The ATX's unique, OmniDirectional movement allows it to travel in all directions thus making it an ideal vehicle to work in tight spaces where turns are not possible and finite control is a necessity. The truck features 48 volt transistor controls with state-of-the-art technology, infinitely variable travel, lift and lower speeds, excellent visibility, ergonomic controls and operator comfort [18].

The unique design of the four 21x12 independently driven Mecanum wheels enables the ATX's OmniDirectional capabilities. Each wheel is directly driven by individual transaxles. The wheels consist of a large, heavy-duty hub with 12 uniquely designed polyurethane rollers. The wheel and roller design provides the Omni-Directional movement of the vehicle based on the speed and direction of each wheel as determined by the operation of the traction joystick. Each roller incorporates bearings that do not require periodic greasing or maintenance under most conditions. Since each roller rotates freely, scrubbing against the floor is minimized while turning or moving sideways [18].



A. 14: Airtrax Sidewinder lift truck [19].

[20] developed an Automated Guided Vehicle as a drive-under tractor in very compact dimensions. The development and realization of the vehicle are optimized for the transportation of small goods (Figure 15.a) [21]. The primary goal was a small vehicle at low cost [22]. Furthermore the vehicle has to be able to transport variable amounts of containers in an economic way. An innovative approach was to accomplish accumulated and single transports by towing a trolley (Figure 15.b) or by carrying one container with the same vehicle. For transport and providing of small goods, the following applications were found to be the most promising: Floor block storage, order picking, assembly and production. The vehicle has an

omnidirectional drive, using four independently and electrically driven Mecanum wheels. The accomplished prototype is smaller than any vehicle that is available at the market in Europe. As a result the needed space for logistic operations, like the width of the track and stations, could be minimized compared to common solutions. Especially the height of the vehicle is very low so that an efficient use as a drive-under tractor is possible. The results showed an efficient approach for an automated transportation of trailers and small load carriers.



A. 15: a) Vehicle with small goods container; b) Vehicle with trolley [20].

### Medical field

Powered wheelchairs are known to provide benefits for older adults by enabling them to have a means of independent mobility. These benefits include: participation in self-care, productivity, and leisure occupations; as well as, socialization opportunities, and positive self worth [23] [24]. Overall powered wheelchairs are linked to an improved quality of life for older adults who have a reduced ability to walk and do not have the stamina, strength, or ability to propel themselves in a manual wheelchair [23]. Without a powered wheelchair these older adults would be dependent on others to complete life tasks [25], and unable to have independent mobility.

The OMNI (Office Wheelchair for High Manoeuvrability and Navigational Intelligence for People with Severe Handicap) is a standalone wheelchair developed with two goals in mind: 1) to allow high mobility in complex environments; and 2) to have modes of operation that will help the user have higher degrees of independence [26]. This wheelchair has been designed for individuals with severe mental and physical disabilities. It consists of Mecanum wheels that provide 3-DOF (degrees of freedom) for the wheelchair; a specialized joystick for 3-DOF movement; a sensor ring around the wheelchair that has IR (infrared) and ultrasound sensors to provide obstacle detection capabilities; a bumper sensor for fail-safe detection of collisions; wheel odometers for knowledge of the wheelchair's location; an elevating seat to raise the user; and a specialized display for the user select modes of operation (Figure 16).

The omnidirectional wheelchair being developed at the University of Western Australia's Centre for Intelligent Information Processing Systems (CIIPS) allows the user to easily manoeuvre in what would otherwise be an extremely complicated environment. This project made improvements to the Mecanum wheels, batteries, motor driver cards, human interface, control software, chassis and suspension system. These improvements transformed the partially working prototype into a fully usable wheelchair (see Figure 16). The result is much higher driving accuracy and a greatly improved overall experience for the user in both comfort and ease of use. On the whole, the project was extremely successful and will provide a very solid test bed for advanced driving and mapping projects in the future [19].



A. 16: Omnidirectional wheelchairs: OMNI [26], CIIPS wheelchair [19], iRW [27] (from left to right).

Another example of an omnidirectional wheelchair is iRW [27], which provides a telehealth system with easy-to-wear, non-invasive devices for real time vital sign monitoring and long-term health care management for the senior users, their family and caregivers (Figure 16). A joystick controller is used to control the iRW to move forward/backward, sway right/left, and spin clockwise/counter clockwise. The maximum forward speed of the iRW is set at 3km/h, which is close to walking speed of human, and the maximum backward and sideways speed is set at 1.5km/h.

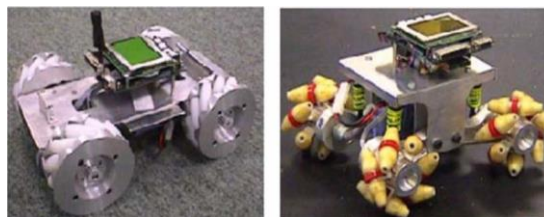
### Educational field

Uranus (Figure 17) was the first mobile robot with Mecanum wheels, designed and constructed in Carnegie Mellon University [28], [29]. It was built to provide a general purpose mobile base to support research in to indoor robot navigation. As a base, it provides full mobility, along with support for a variety of payloads, such as sensors and computers. It had not a suspension system, which is absolutely necessary if the ground is not completely flat.



A. 17: Uranus omnidirectional mobile robot [28].

Other researchers, such as Braunl from University of South Australia have developed two different Mecanum wheel omnidirectional mobile robots, Omni-1 and Omni-2 [30], [31]. Figure 18 shows the structure of Omni-1 and Omni-2. The first design, Omni-1 used the Mecanum wheel design with rims that only leave a small gap/clearance for the roller. The motor and wheel assembly tightly attached to robot's chassis. The Omni-1 can drive very well on hard and flat surface but it loses the omnidirectional capability on soft surface.

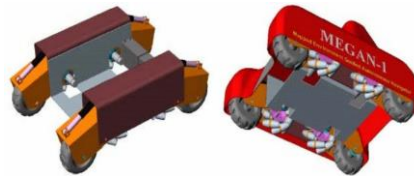


A. 18: Mecanum Wheel Mobile robot Omni-1 and Omni-2 from University of Western Australia [31].



Omni-2 was developed using rimless and with centrally mounted roller. The motor and wheel assembly attach to cantilever wheel suspension with shock absorbers. The rimless Mecanum wheel and shocks absorbers encounter the sinking-in on softer surface and uneven work surface as a result allows omnidirectional driving for Omni-2.

The Mechatronics and Robotics Research Group (MR2G) at Massey University have developed all terrain Automatic Guided Vehicle (AGV) using a set of Mecanum wheels combined with a set of conventional wheels [11]. Any terrain change is automatically detected and a set of pneumatics actuators used to change from Mecanum wheels for indoor and high mobility requirement to conventional wheel for outdoor and rough terrain. This new driving mechanism of AGV has been implemented on Mapped Environment Guided Autonomous Navigator (MEGAN). Figure 19 illustrates the structure of MEGAN.



A. 19: Mapped Environment Guided Autonomous Navigator (MEGAN) [11].

#### Other fields

The main goal of the CommRob project was to develop scientific methods or technologies to introduce robots in human environments. The Interactive Behavior Operated Trolley (Figure 20) InBOT addresses several everyday problems. Among other possibilities this means helping the customer to find the desired products without extensive search in big supermarkets, or relieving the customer from the burden of pushing the shopping cart using his own force all the time, especially if the cart is heavily loaded or the customer is elderly or handicapped. Especially for these groups of customers it could be very interesting that InBOT is able to avoid collisions on its own, even with objects that are moving themselves. InBOT has the ability to perform special local maneuvers and a flexible task-planner. It provides four different modes of operation to assist the user in the best way. First InBOT can be steered like an ordinary shopping cart by the haptic handle [32], including assistance functionalities like obstacle avoidance. Second and third it can follow or lead the user. Therefore the robot has to continuously track the user's position, to perform adaptive distance management and finally to estimate the user's intentions. And fourth the robot can be commanded to act independently until ordered otherwise [33].



A. 20: The interactive shopping trolley [33].



## Conclusions

In this paper, an overview over the Mecanum wheels and their practical applications is presented. The main advantage of this type of wheel is represented by the omnidirectional property that it provides, allowing extreme maneuverability and mobility in congested environments. Also, some research that was carried out in Mecanum wheel mobile robots in order to improve the wheel design is described. The manoeuvrability provided by omnidirectional vehicles can be utilized and can be very important in both outdoors applications, such as search and rescue missions, military activities, planetary explorations and mine operations, long loads transportation, and indoor applications, like small goods transportation, powered robotic wheelchairs or shopping carts.

## Acknowledgement

This paper was realized with the support of POSDRU CUANTUMDOC “DOCTORAL STUDIES FOR EUROPEAN PERFORMANCES IN RESEARCH AND INOVATION” ID79407 project, funded by the European Social Found and Romanian Government.

## Bibliography

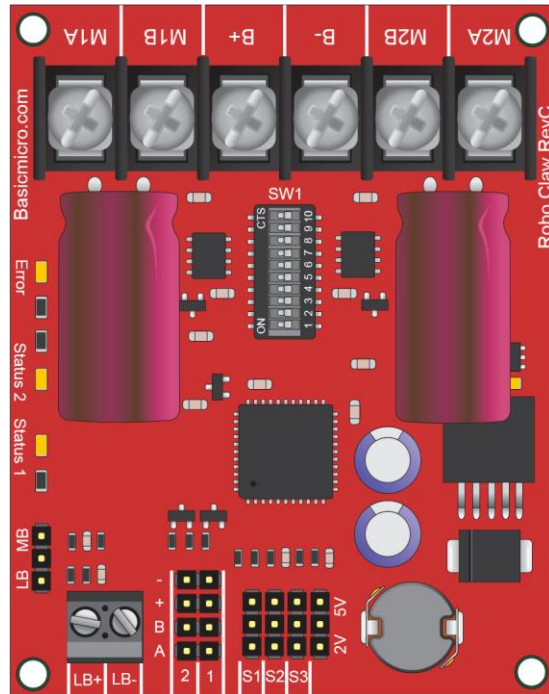
- [1] Ilon, B.E., ‘Wheels for a course stable self propelling vehicle movable in any desired direction on the ground or some other base’, US Patent and Trademarks office, Patent 3.876.255, 1975.
- [2] Song, J.B., Byun, K.S., “Design and Control of a Four-Wheeled Omnidirectional Mobile Robot with Steerable Omnidirectional Wheels”, *Journal of Robotic Systems*, 21(4), 2004, pp. 193-208.
- [3] Doroftei, I., Stirbu, B., “Design, Modeling and Control of an Omni-directional Mobile Robot”, *Solid State Phenomena Vols. 166-167*, 2010, pp 173-178, Trans Tech Publications, Switzerland. doi:10.4028/www.scientific.net/SSP.166-167.173.
- [4] Dickerson, S.L., Lapin, B.D., „Control of an omni-directional robotic vehicle with Mecanum wheels”, in *National Tele-systems Conference Proceedings*, p. 323-328, March 26-27, Atlanta, USA, 1991.
- [5] Borenstein, J., Everett, H.R., Feng, L., „Navigating Mobile Robots: Sensors and Techniques”, A K Peters, Ltd, MA, USA, 1996.
- [6] Doroftei, I., Grosu, V., Spinu, V., „Omnidirectional Mobile Robot – Design and Implementation”, *Bioinspiration and Robotics: Walking and climbing Robots*, Book edited by: Maki K. Habib, ISBN 978-3902613-15-8, pp. 544, I-Tech, Vienna, Austria, EU, September 2007.
- [7] Nagatani, K., Tachibana, S., Sofne, M., Tanaka, Y., „Improvement of odometry for omnidirectional vehicle using optical flow information”, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.
- [8] Everett, H.R., „Sensors for Mobile Robots: Theory and Application”, A K Peters, Ltd, MA, USA, 1995.
- [9] Giachetti, A., Campani, M., Torre, V., „The use of optical flow for road navigation”, *IEEE Transactions on Robotics and Automation*, Vol.14, No.1, pp. 34-48, 1998.
- [10] Doroftei, I., Grosu, V., Spinu, V., ”Design aspects of omnidirectional wheels and mobile platforms”, *Proceedings of the 4<sup>th</sup> International Conference on Robotics*, Buletin of the Transylvania University of Brasov, Vol. 15(50), Series A, Special Issue, ISSN 1223-9631, 2008.
- [11] Diegel, O., Badve, A., Bright, G., Potgieter, J., Tlale, S., „Improved Mecanum Wheel Design for Omnidirectional Robots”, *Proc. Australasian Conference on Robotics and Automation*, Auckland, November 27-29, pp. 117-121, 2002.
- [12] Ramirez-Serrano, A., Kuzyk, R., „Modified Mecanum Wheels for Traversing Rough Terrain”, 6<sup>th</sup> *International Conference on Autonomic and Autonomous Systems*, IEEE Computer Society, 2010, doi: 10.1109/ICAS.2010.35
- [13] Lippit, T.C., Jones, W.C., „OmniBot Mobile Base”, *KSC Re-search and Technology Report*, NASA, USA, 1998.

- [14] <http://rtreport.ksc.nasa.gov/techreports/98report/09-ar/ar06.html>
- [15] <http://www.robotics.com/robomenu/odv.html>
- [16] <http://www.architectureandvision.com>
- [17] Ransom, S., Krömer, O., Lückemeier, M., "Planetary rovers with Mecanum wheels", 16<sup>th</sup> ISTVS Intl Conf, Nov 25-28, 2008, Torino, Italy.
- [18] [http://www.lomag-man.org/nouveautes/20vehicules/airtrax\\_omnidirectionnel.pdf](http://www.lomag-man.org/nouveautes/20vehicules/airtrax_omnidirectionnel.pdf)
- [19] Tuck-Voon How, „Development of an Anti-Collision and Navigation System for Powered Wheelchairs”, 2010
- [20] Schulze, L., Behling, S., Buhrs, S., „Development of a Micro Drive-Under Tractor - Research and Application”, Proceedings of the International MultiConference of Engineers and Computer Scientists, Vol. II, March 16-18, 2011, Hong-Kong.
- [21] Furmans, K., Schleyer, M., Schönung, F., “A Case for Material Handling Systems, Specialized on Handling Small Quantities”, in Proc. of 10<sup>th</sup> International Material Handling Research Colloquium, Dortmund, Germany, 2008, pp. 185-197.
- [22] Schulze, L., Behling, S., Buhrs, S., “Automated Guided Vehicle Systems: a Driver for Increased Business Performance”, in Proc. of International Multi Conference of Engineers and Computer Scientists 2008, Hong Kong, pp. 1275-1280.
- [23] Brandt, A., Iwarsson, S., Stahle, A., "Older people's use of powered wheelchairs for activity and participation", J. Rehabil. Med., no. 36, pp. 70-77, 2004.
- [24] Evans, R., "The effect of electrically powered indoor/outdoor wheelchairs on occupation: a study of users' views", British Journal of Occupational Therapy, vol. 63, no. 11, pp. 547-553, 2000.
- [25] Hardy, P., "Powered wheelchair mobility: An occupational performance evaluation perspective", Australian Occupational Therapy Journal, no. 51, pp. 34-42, 2004.
- [26] Hoyer, H., Borgolte, U., Jochheim, A., "The OMNI-Wheelchair - State of the art," in Center on Disabilities, Technology and Persons with Disabilities Conference, Northridge, CA, 1999.
- [27] Po-Er Hsu, Yeh-Liang Hsu, Jun-Ming Lu, Jerry, J.-H. Tsai, Yi-Shin Chen, „iRW: An Intelligent Robotic Wheelchair Integrated with Advanced Robotic and Telehealth Solutions”, in „1<sup>st</sup> Asia Pacific eCare and TeleCare Congress,” June 16-19, 2011, Hong Kong, China.
- [28] Muir, P. F., Neuman, C.P., „Kinematic Modeling for Feedback Control of an Omnidirectional Wheeled Mobile Robot”, IEEE International Conference on Robotics and Automation, 1987.
- [29] Muir, P. F., Neuman, C. P., „Kinematic Modeling of Wheeled Mobile Robots”, Journal of Robotic Systems, No. 4(2), pp. 281-340, 1987.
- [30] Braunl, T., „Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems”, First Edition. Springer-Verlag, Berlin, 2003.
- [31] <http://robotics.ee.uwa.edu.eyebot/omni.html>
- [32] Goller, M., Kerscher, T., Ziegenmeyer, M., Ronnau, A., Zollner, J. M., Dillmann, R., „Haptic Control for the Interactive Behavior Operated shopping Trolley InBOT”, in New Frontiers in Human-Robot Interaction symposium at the Artificial Intelligence and Simulation of Behaviour (AISB) 2009 Convention, 2009.
- [33] Goller, M., Kerscher, T., Zollner, J.M., Dillmann, R., Devy, M., Germa, T., Lerasle, F., „Setup and Control Architecture for an Interactive Shopping Cart in Human All Day Environments”, In Proceeding of the Int. Conf. on Advanced Robotics (ICAR'09), pp. 1-6, 2009.

## Appendix B: RoboClaw 2 Channel 15A Motor Controller Data Sheet [52]

### Feature Overview:

- 2 Channel at 15Amp, Peak 30Amp
- Battery Elimination Circuit (BEC)
- Switching Mode BEC
- Hobby RC Radio Compatible
- Serial Mode
- TTL Input
- Analog Mode
- 2 Channel Quadrature Decoding
- Thermal Protection
- Lithium Cut Off
- Packet Serial
- High Speed Direction Switching
- Flip Over Switch
- Over Current Protection
- Regenerative Braking



### Basic Description

The RoboClaw 2X15 Amp is an extremely efficient, versatile, dual channel synchronous regenerative motor controller. It supports dual quadrature encoders and can supply two brushed DC motors with 15 Amps continuous and 30 Amp peak.

With dual quadrature decoding you get greater control over speed and velocity. Automatically maintain a speed even if load increases. RoboClaw also has a built in PID routine for use with an external control system.

RoboClaw is easy to control with several built in modes. It can be controlled from a standard RC receiver/transmitter, serial device, microcontroller or an analog source, such as a potentiometer based joystick. RoboClaw is equipped with screw terminal for fast connect and disconnect. All modes are set by the onboard dip switches making setup a snap!

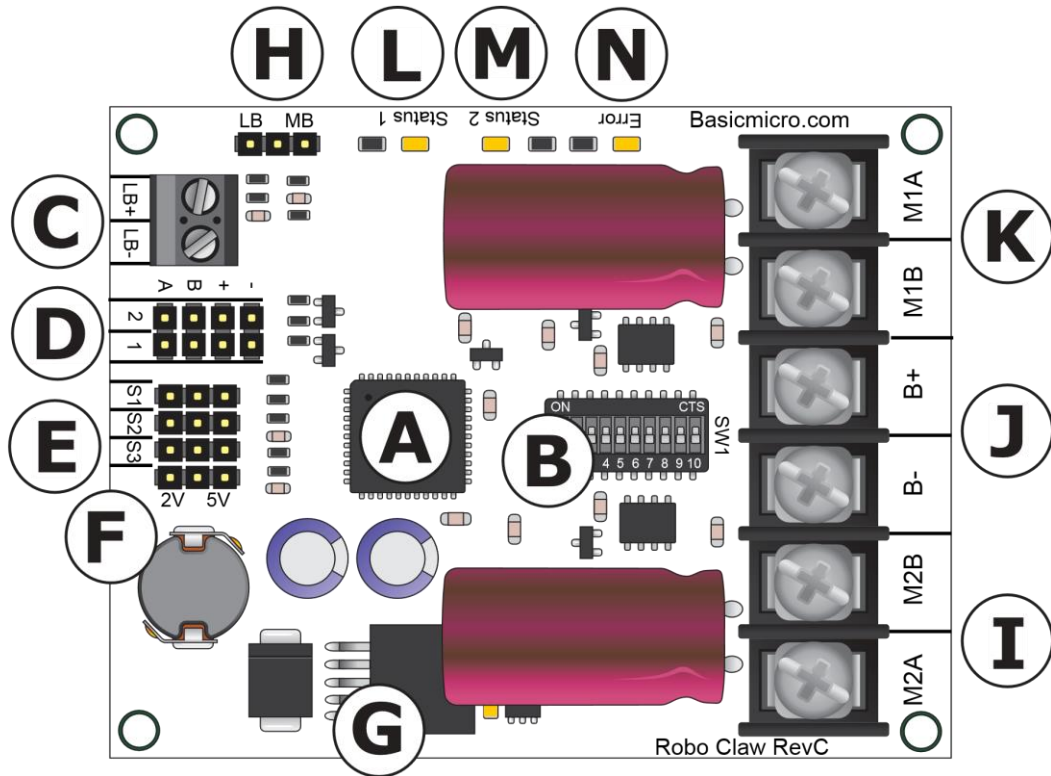
### Optical Encoders

RoboClaw features dual channel quadrature decoding. RoboClaw gives you the ability to create a closed loop system. Now you can know a motors speed and direction giving you greater control over DC motors systems.

## Power System

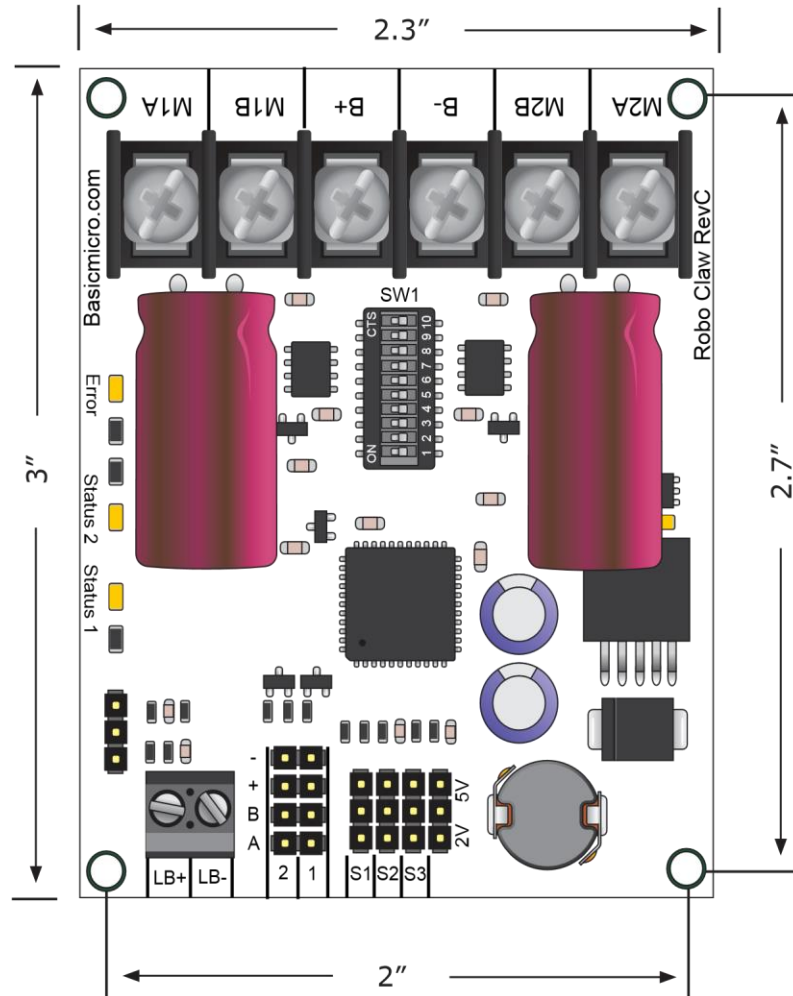
The RoboClaw is equipped with synchronous regenerative motor drivers. This means your battery is recharged when slowing down, braking or reversing. In addition a switching mode BEC is included. It can supply a useful current of up to 3Amps. The BEC is meant to provide power to a microcontroller or RC receiver.

## Hardware Overview:



- A:** Microcontroller
- B:** DIP Switch
- C:** Logic Battery 3.5mm Screw Terminals
- D:** Encoder Input Header
- E:** Input Control Headers
- F:** VCC on Input Control Header Disable Jumper
- G:** Logic Voltage Regulator
- H:** Logic Voltage Source Selection Header
- I:** DC Motor Channel 2 Screw Terminals
- J:** Main Battery Input Screw Terminals
- K:** DC Motor Channel 1 Screw Terminals
- L:** Status LED 1
- M:** Status LED 2
- N:** Error LED

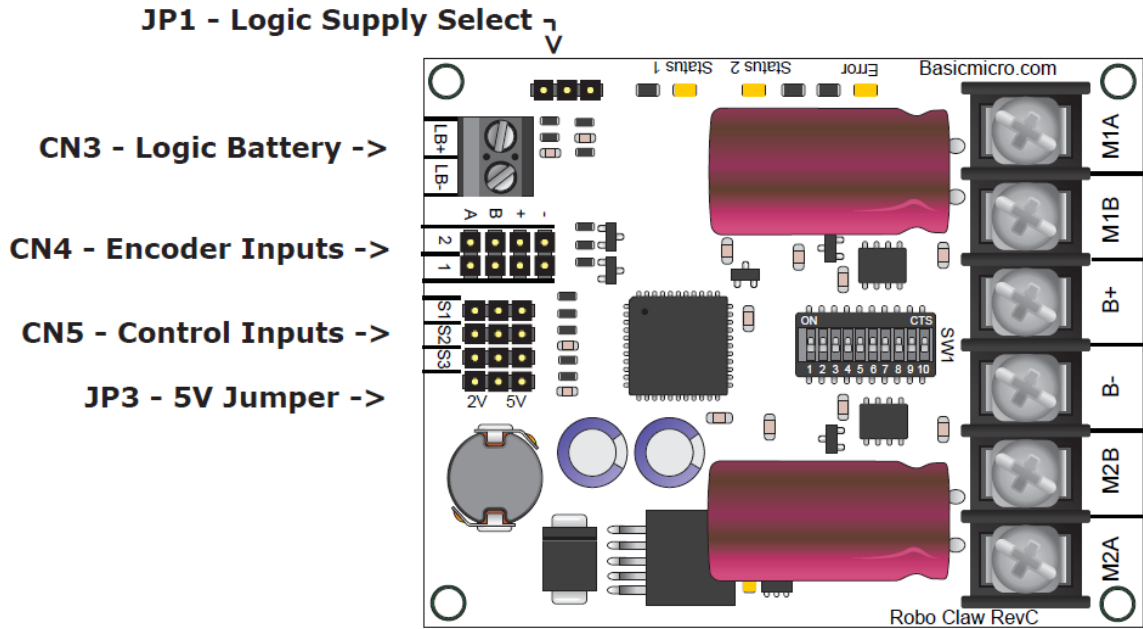
**Dimensions:**



**Board Edge:** 2.3"W X 3"L

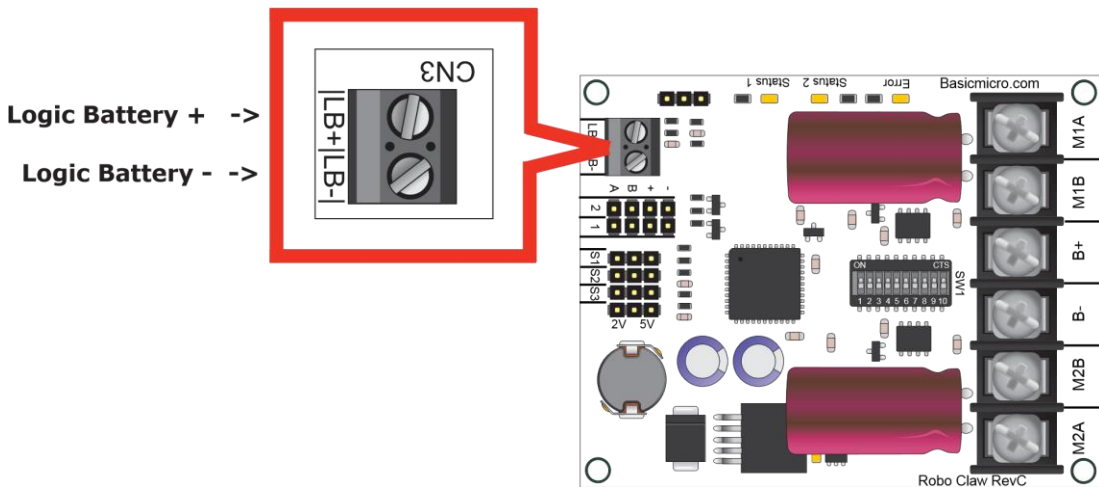
**Hole Pattern:** 0.125D, 2"W x 2.7"H

## Header Overview



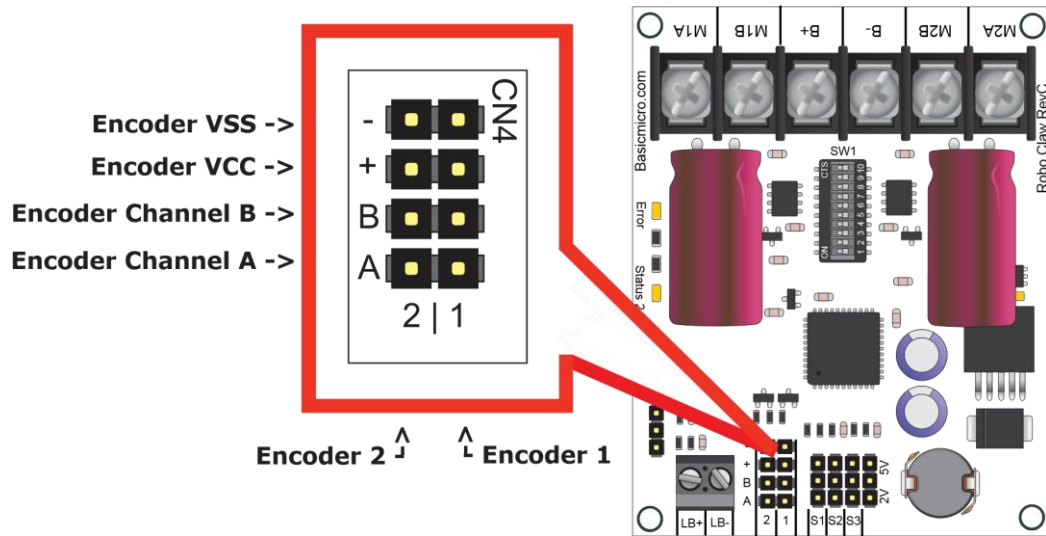
### Logic Battery Screw Terminals

The logic circuits can be powered from the main battery or a secondary battery wired to CN3. JP1 controls what source powers the logic circuits. The maximum input voltage for the logic supply is 30VDC.



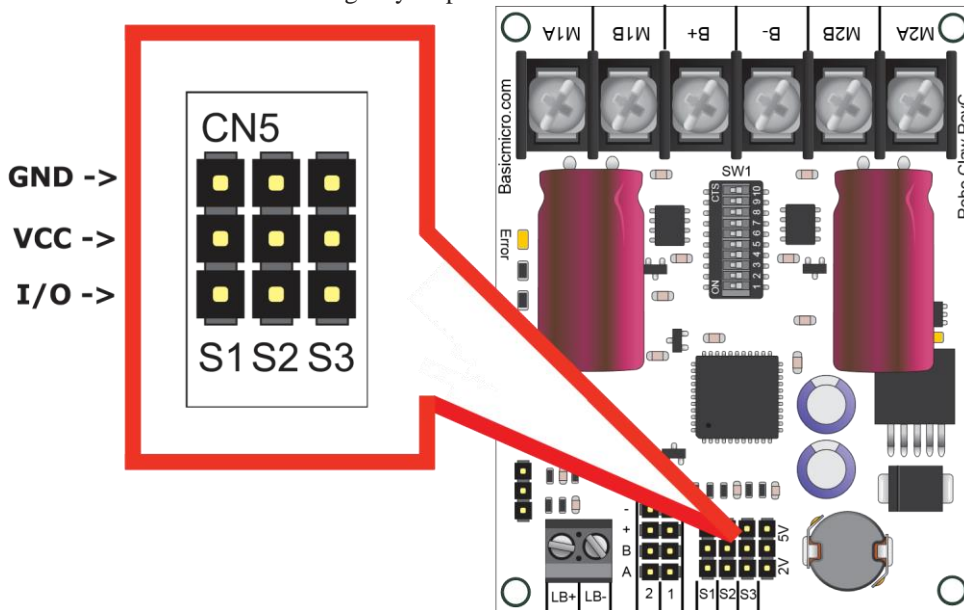
### Encoder Inputs

This header is setup for dual quadrature encoders. A and B are the inputs from the encoders. The header also supplies 5VDC to power the encoders. When connecting the encoder make sure the leading channel for the direction of rotation is connected to A. If one encoder is backwards to the other you will have one internal counter counting up and the other counting down. Which will affect the operation of Robo Claw. Refer to the data sheet of the encoder you are using for channel direction.



### Control Inputs

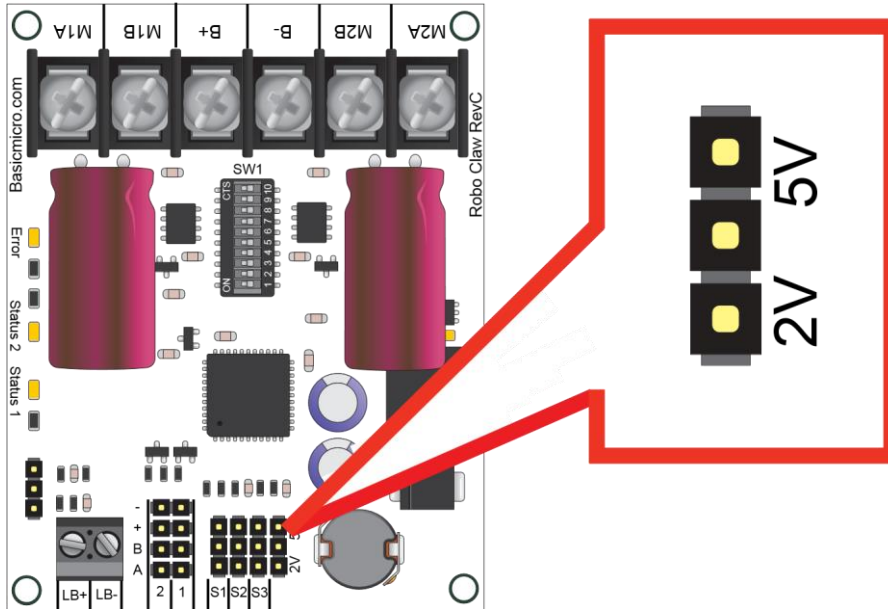
S1, S2 and S3 are setup for standard servo style headers GND, +5V and I/O. S1 and S2 are the control inputs for serial, analog and RC modes. S3 used as a flip switch input when in RC or Analog modes. In serial mode S3 becomes a emergency stop.





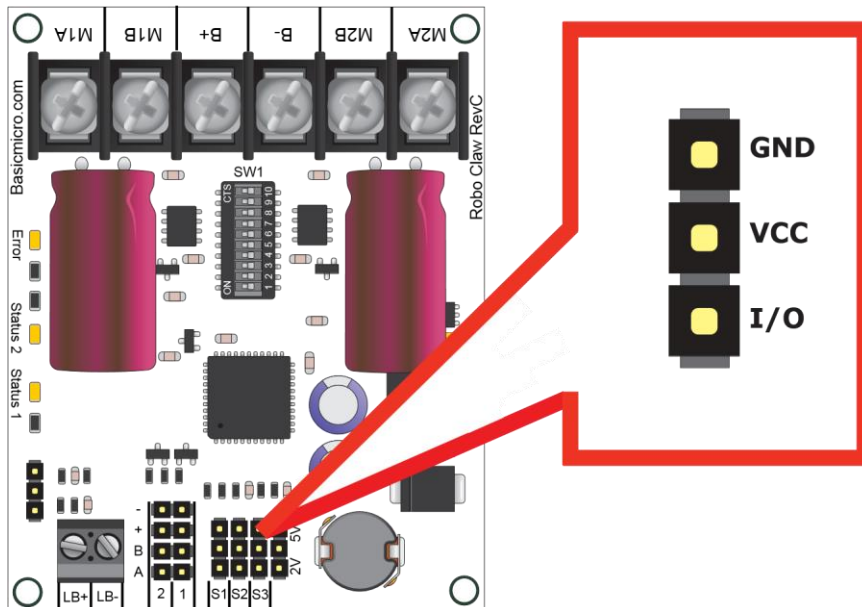
### BEC Jumper

VCC on control input headers S1,S2 and S3 can be turned off, on or 2V by the jumper near S3 on CN5. Removing the BEC jumper disables VCC on S1, S2 and S3 headers. In some systems the RC receiver may have its own supply and will conflict with the RoboClaw logic supply.



### Emergency Stop

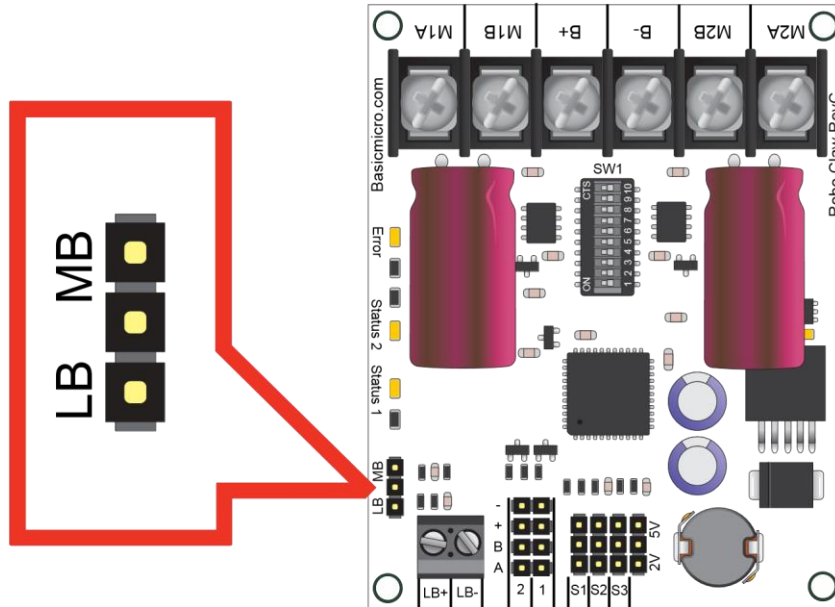
In serial mode S3 becomes the emergency stop. S3 is active low. It is internally pulled up so it will not accidentally trip.





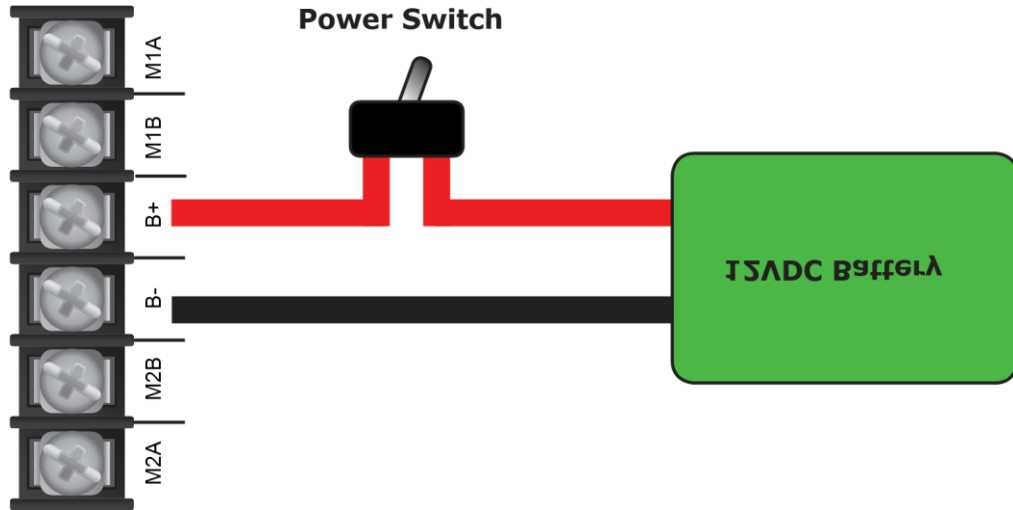
### Logic Supply Select

The RoboClaw logic requires 5VDC which is provided from the on board regulator. The regulator source input is set with the logic supply jumper. Set to LB for a separate logic battery or MB for the main battery as the source.



### Main Battery Screw Terminals

The main battery connections are marked with a B- and B+ on the main screw terminal. B+ is the positive side of the battery typically marked with a red wire. The B- is the negative side of the battery and typically marked with a black wire. When connecting the main battery its a good practice to use a switch to turn the main power on and off. When placing a switch in between the RoboClaw and main battery you must use a switch with the proper current rating. Since the RoboClaw can draw up to 30Amps peak you should use a switch rated for at least 40Amps. The main battery can be 6V to 30V DC.



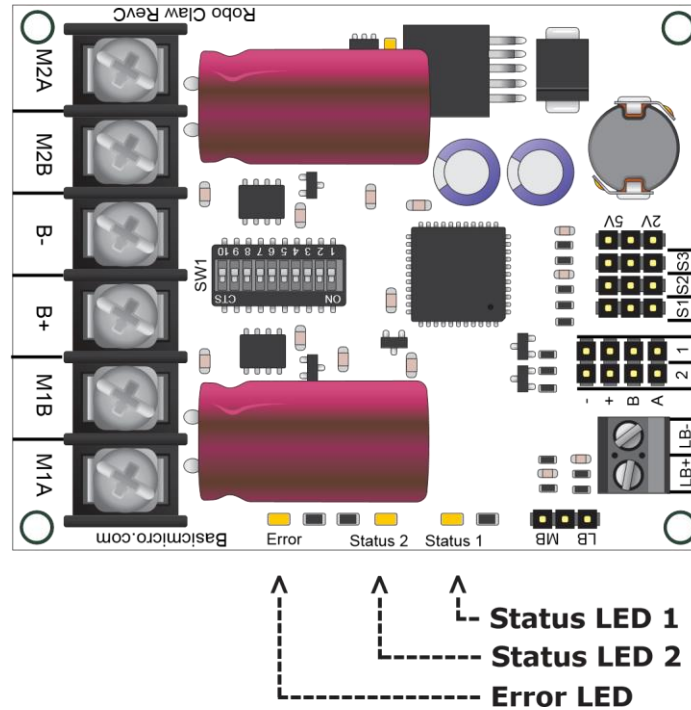
### Motor Screw Terminals

The motor screw terminals are marked with M1A / M1B for channel 1 and M2A / M2B for channel 2. There is no specific polarities for the motors. However if you want both motors turning in the same direction on a 4 wheeled robot you need to reverse one of the motors as shown below:



### Status and Error LEDs

The RoboClaw has 3 main LEDs. 2 Status LEDs and 1 Error LED. When Robo Claw is first powered up all 3 LED should blink several times briefly to indicate all 3 LEDs are functional. The status LEDs will indicate a status based on what mode RoboClaw is set to.



#### Analog Mode

Status LED 1 = On continuous.

Status LED 2 = On when motor(s) active.

#### RC Mode

Status LED 1 = On continuous, blink when pulse received.

Status LED 2 = On when motor(s) active.

#### Serial Modes

Status LED 1 = On continuous, blink on serial receive.

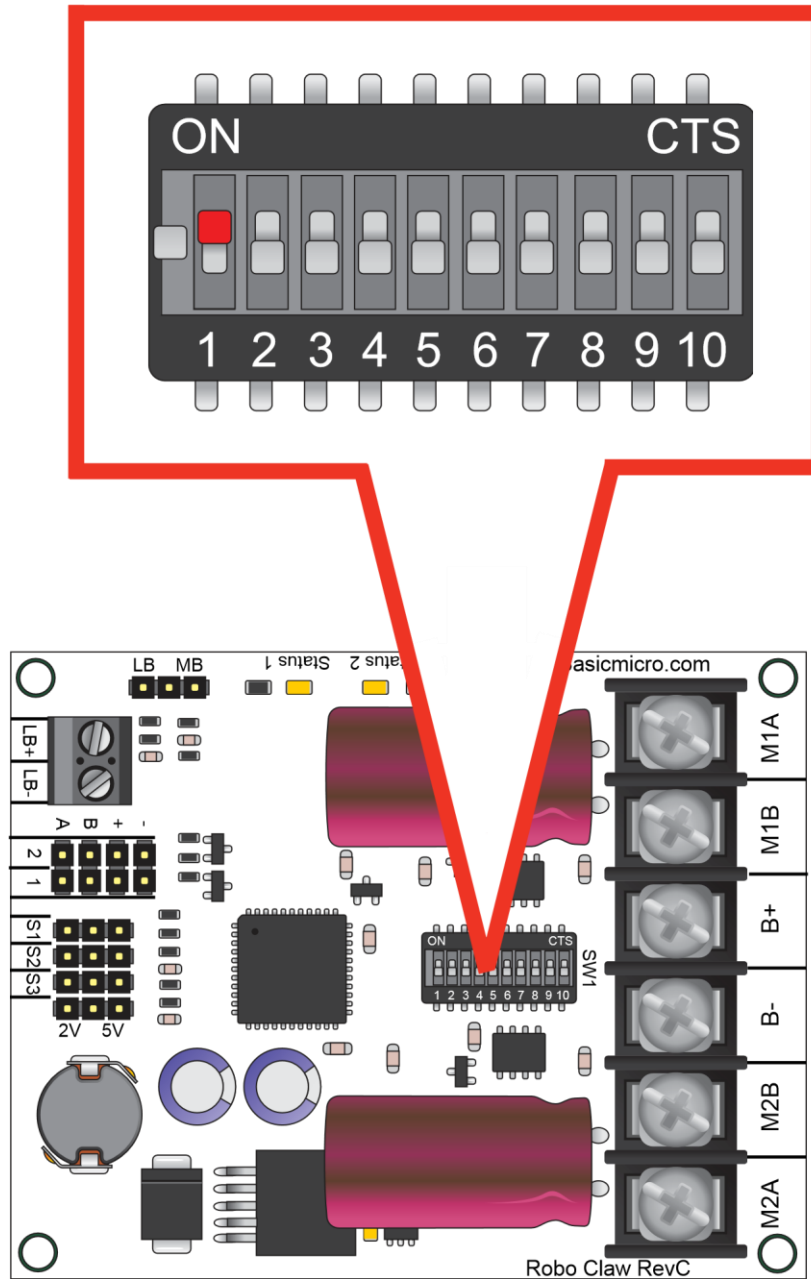
Status LED 2 = On when motor(s) active.

#### Errors

- |                 |  |
|-----------------|--|
| Over Current    | = Error LED on solid. Status 1 or 2 indicates which motor. |
| Over Heat       | = Error LED blinking once with a long pause.               |
| Main Batt Low   | = Error LED blinking twice with a long pause.              |
| Main Batt High  | = Error LED on fast flicker until condition is cleared.    |
| Logic Batt Low  | = Error LED blinking three times with a long pause.        |
| Logic Batt High | = Error LED blinking four times with a long pause.         |

### DIP Switch Overview

The dip switch on RoboClaw is used to set its operating modes and the many options. The switch is marked with an ON label at its top. The switches are also labeled from left to right starting with switch 1 and ending with switch 10. When a dip switch is moved toward the label ON it is considered ON. When the switch is facing away from the ON label it is considered off. Be careful to ensure the switch is not floating in between and is firmly OFF or ON. See illustration below. The red switch (SW1) is in the ON position. The grey colored switches are in the OFF position.



### Low Voltage Cutoff

RoboClaw has a built in low voltage protection. This has two main purposes. To protect RoboClaw from running erratically when the main battery level gets to low and protect a Lithium battery from being damaged.

Voltage	SW8	SW9	SW10
Not Monitored	OFF	OFF	OFF
Lead Acid - Auto	ON	OFF	OFF
2- Cell (6V Cutoff)	OFF	ON	OFF
3- Cell (9V Cutoff)	ON	ON	OFF
4- Cell (12V Cutoff)	OFF	OFF	ON
5- Cell (15V Cutoff)	ON	OFF	ON
6- Cell (18V Cutoff)	OFF	ON	ON
7- Cell (21V Cutoff)	ON	ON	ON

### **RoboClaw Modes**

There are 4 modes. Each with a specific way to control RoboClaw. The following list explain each mode and the ideal application.

#### **Mode 1 - RC Input**

With RC input mode RoboClaw can be controlled from any hobby RC radio system. RC input mode also allows low powered microcontroller such as a Basic Stamp or Nano to control RoboClaw. RoboClaw expects servo pulse inputs to control the direction and speed. Very similar to how a regular servo is controlled. RC mode can not use encoders.

#### **Mode 2 - Analog**

Analog mode uses an analog signal from 0V to 5V to control the speed and direction of each motor. RoboClaw can be controlled using a potentiometer or filtered PWM from a microcontroller. Analog mode is ideal for interfacing RoboClaw joystick positioning systems or other non microcontroller interfacing hardware. Analog mode can not use encoders.

#### **Mode 3 - Simple Serial**

In simple serial mode RoboClaw expects TTL level RS-232 serial data to control direction and speed of each motor. Simple serial is typically used to control RoboClaw from a microcontroller or PC. If using a PC a MAX232 type circuit must be used since RoboClaw only works with TTL level input. Simple serial includes a slave select mode which allows multiple RoboClaws to be controlled from a signal RS-232 port (PC or microcontroller). Simple serial is a one way format, RoboClaw only receives data.

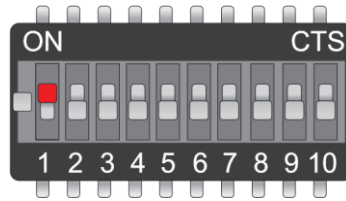
#### **Mode 4 - Packet Serial**

In packet serial mode RoboClaw expects TTL level RS-232 serial data to control direction and speed of each motor. Packet serial is typically used to control RoboClaw from a microcontroller or PC. If using a PC a MAX232 type circuit must be used since RoboClaw only works with TTL level input. In packet serial mode each RoboClaw is assigned an address using the dip switches. There are 8 addresses available. This means up to 8 RoboClaws can be on the same serial port. When using the quadrature decoding feature of RoboClaw packet serial is required since it is a two way communications format. This allows RoboClaw to transmit information about the encoders position and speed.

# RC Input

### Mode 1 - RC Input

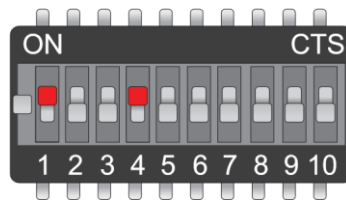
For RC mode set SW1 = ON. RC mode is typically used when controlling RoboClaw from a hobby RC radio. This mode can also be used to simplify driving RoboClaw from a microcontroller using servo pulses. There are 4 options in RC Input mode. These options are set with SW4, SW5, SW6 and SW7.



### Switch 4 - Mixing Mode

SW4 = ON: Turns mixing mode ON. S1 controls forward and reverse. S2 controls steering. Control will be like a car.

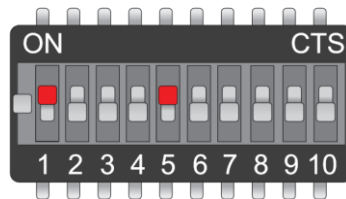
SW4 = OFF: Turns mixing mode OFF. S1 controls motor 1 speed and direction. S2 controls motor 2 speed and direction. Control will be like a tank.



### Switch 5 - Exponential Mode

SW5 = ON: Turns exponential mode ON. Exponential response softens the center control position. This mode is ideal with tank style robots.

SW5 = OFF: Turns exponential mode OFF. Motor response will be linear and directly proportional to the control input. Ideal for 4 wheel style robots.

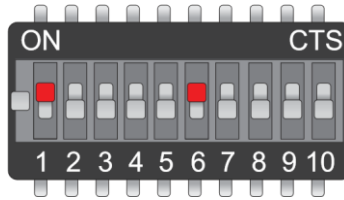




**Switch 6 - MCU or RC Control**

SW6 = ON: Turns MCU control mode ON. RoboClaw will continue to execute last pulse received until new pulse received. Signal lost fail safe and auto calibration are off in this mode.

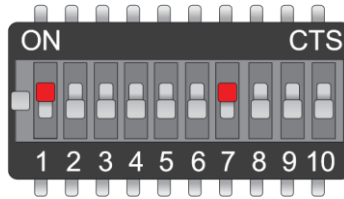
SW6 = OFF: Turns RC control mode ON. RoboClaw will calibrate the center and end points automatically to maximize stick throw. This mode includes a fail safe. If control input is lost, RoboClaw will shut down.



**Switch 7 - Flip Switch Input**

SW7 = ON: Flip switch input requires servo pulse. Pulse greater than 1.5ms will reverse steering control. The flip switch is typically used in robot combats to automatically reverse the controls if a robot is flipped over.

SW7 = OFF: Flip switch input expects TTL control signal. 0V for flipped and 5V for normal.



### Servo Pulse Ranges

The RoboClaw expects RC servo pulses on S1 and S2 to drive the motors when the dip switches are set for RC mode. The center points are calibrated at start up. 1000us is the default for full reverse and 2000us is the default for full forward. The RoboClaw will auto calibrate these ranges on the fly. If a pulse smaller than 1000us or larger than 2000us is detected the new pulses will be set as the new range.

Pulse	Function
1000us	Full Reverse
2000us	Full Forward



### RC Control - Arduino Example

The example will drive a 2 motor 4 wheel robot in reverse, stop, forward, left turn and then right turn. The program was written and tested with a Arduino Uno and P5 connected to S1, P6 connected to S2. Set switches SW1, SW4, SW5 and SW6 to ON.

```
//Basic Micro Robo Claw RC Mode. Control Robo Claw //with servo pulses from a
microcontroller.
//Switch settings: SW1=ON, SW4=ON, SW5=ON and SW6=ON

#include <Servo.h>

Servo myservo1; // create servo object to control a Roboclaw channel Servo myservo2; // create
servo object to control a Roboclaw channel

int pos = 0; // variable to store the servo position

void setup()
{
  myservo1.attach(5); // attaches the RC signal on pin 5 to the servo object myservo2.attach(6);
// attaches the RC signal on pin 6 to the servo object
}

void loop()
{
  myservo1.writeMicroseconds(1500); //Stop myservo2.writeMicroseconds(1500); //Stop
delay(2000);

  myservo1.writeMicroseconds(1250); //full forward delay(1000);

  myservo1.writeMicroseconds(1500); //stop delay(2000);

  myservo1.writeMicroseconds(1750); //full reverse delay(1000);

  myservo1.writeMicroseconds(1500); //Stop delay(2000);

  myservo2.writeMicroseconds(1250); //full forward delay(1000);

  myservo2.writeMicroseconds(1500); //Stop delay(2000);

  myservo2.writeMicroseconds(1750); //full reverse
delay(1000);
}
```

### RC Control - BasicATOM Pro Example

The example will drive a 2 motor 4 wheel robot in reverse, stop, forward, left turn and then right turn. The program was written and tested with a BasicATOM Pro and P0 connected to S1, P1 connected to S2. Set switches SW1, SW4, SW5 and SW6 to ON.

```
;Basic Micro Robo Claw RC Mode. Control Robo Claw ;with servo pulses from a
microcontroller.
;Switch settings: SW1=ON, SW4=ON, SW5=ON and SW6=ON

Main
  pulsout P15,(1500*2) ; stop      pulsout P14,(1500*2) ; stop      pause 2000

  pulsout P15,(500*2)           ; full backward
  pause 1000

  pulsout P15,(1500*2) ; stop
  pause 2000

  pulsout P15,(2500*2) ; full forward      pause 1000

  pulsout P15,(1500*2) ; stop
  pause 2000

  pulsout P14,(500*2)           ; left turn
  pause 1000

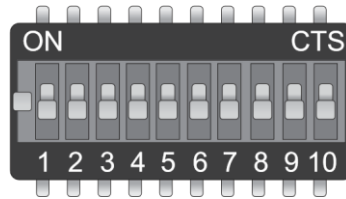
  pulsout P14,(1500*2) ; stop
  pause 2000

  pulsout P14,(2500*2) ; right turn
  pause 1000
goto main
```

# Analog Input

### Mode 2 - Analog Input

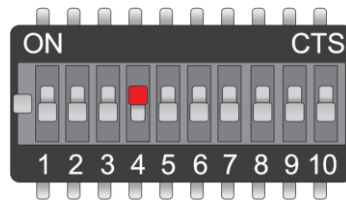
For Analog mode set SW1,SW2 and SW3 = OFF. In this mode S1 and S2 are set as analog inputs. Voltages of 0V = Full reverse, 1V = Stop and 2V = Full forward. You can use linear potentiometers of 1K to 100K to control RoboClaw. Or you can use a PWM signal to control RoboClaw in analog mode. If using a PWM signal to control RoboClaw you will need a simple filter circuit to clean up the pulse. If using a potentiometer set the BEC header to 2V.



### Switch 4 - Mixing Mode

SW4 = ON: Turns mixing mode ON. One channel input to control forward and reverse. Second channel input for steering control. Control will be like a car.

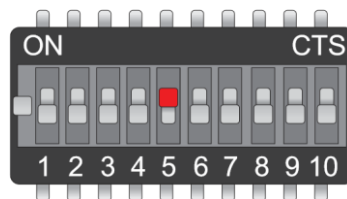
SW4 = OFF: Turns mixing mode OFF. One channel controls one motor speed and direction. Second channel controls the other motor speed and direction. Control will be like a tank.



### Switch 5 - Exponential Mode

SW5 = ON: Turns exponential mode ON. Exponential response softens the center control position. This mode is ideal with tank style robots.

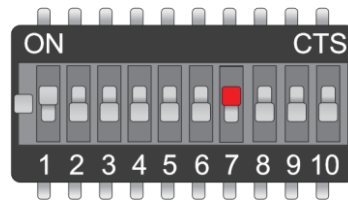
SW5 = OFF: Turns exponential mode OFF. Motor response will be linear and directly proportional to the control input. Ideal for 4 wheel style robots.



### Switch 7 - Flip Switch

SW7 = ON: Turns the flip switch input S3 on. The flip switch signal is a TTL driven signal. 0V is active and 5V is not active. When the flip switch signal is active all inputs to RoboClaw are reversed.

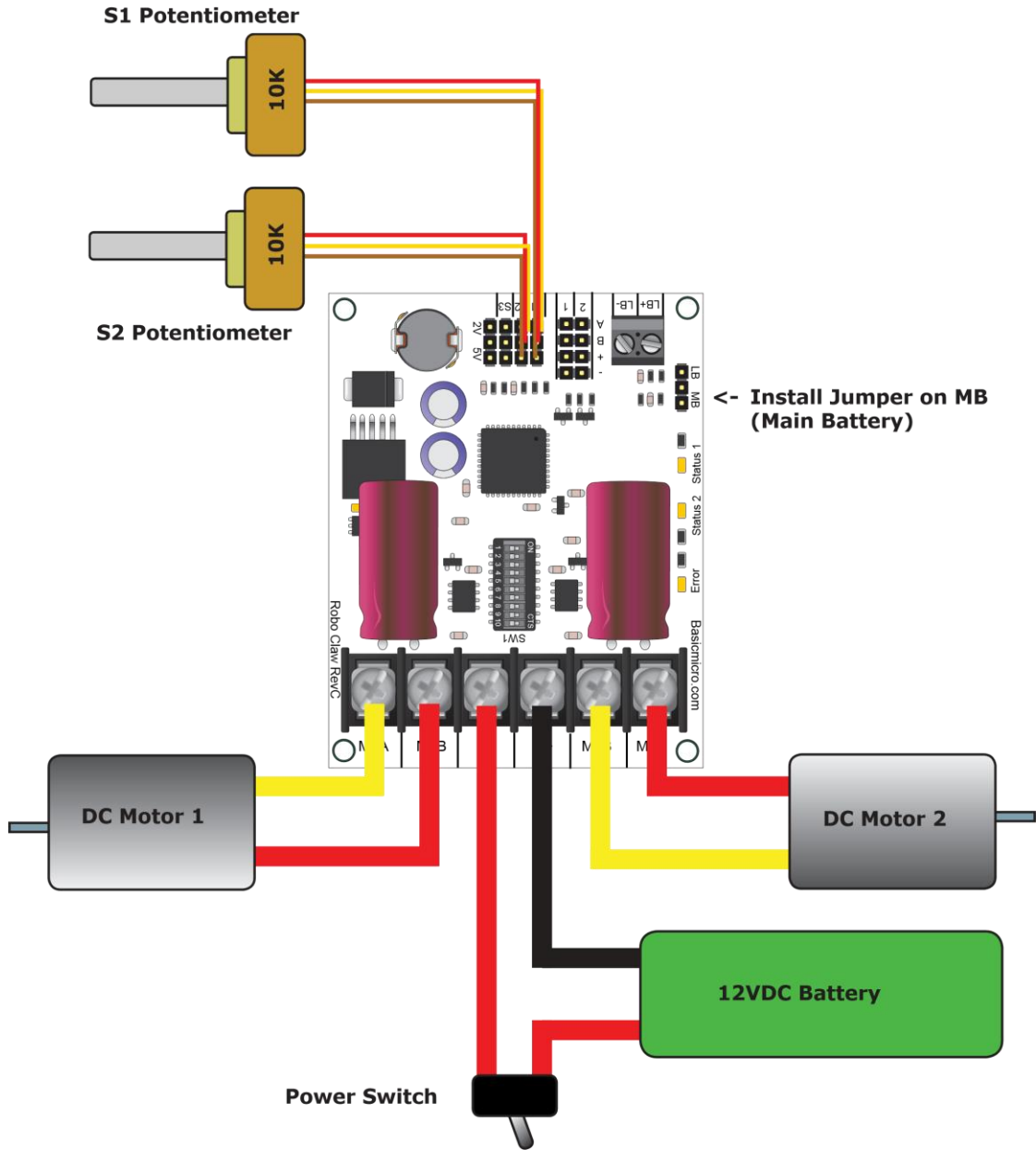
SW7 = OFF: Turns flip switch input S3 off.





### Analog Wiring Example

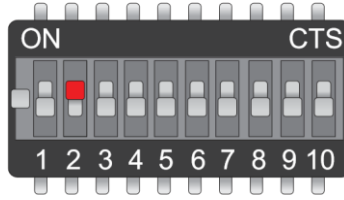
Connect the RoboClaw as shown below using two potentiometers. Install BEC 2V jumper and set switch SW4 to ON (Mixing Mode). You can also use the wire example with SW4 OFF. Center the potentiometers before applying power or the attached motors will start moving. S1 potentiometer in mix mode (SW4) will control forward and reverse. S2 potentiometer in mix mode (SW4) will control turning (LEFT / RIGHT).



# Simple Serial

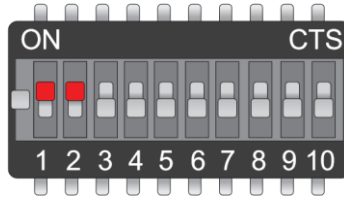
### Mode 3 - Simple Serial

Simple Serial mode set SW1 = OFF, SW2 = ON and SW3 = OFF. In this mode S1 accepts TTL level byte commands. RoboClaw is receive only and uses 8N1 format which is 8 bits, no parity bits and 1 stop bit. If your using a microcontroller you can interface directly to RoboClaw. If your using a PC a level shifting circuit is required (MAX232). The baud rate is set by the dip switches.



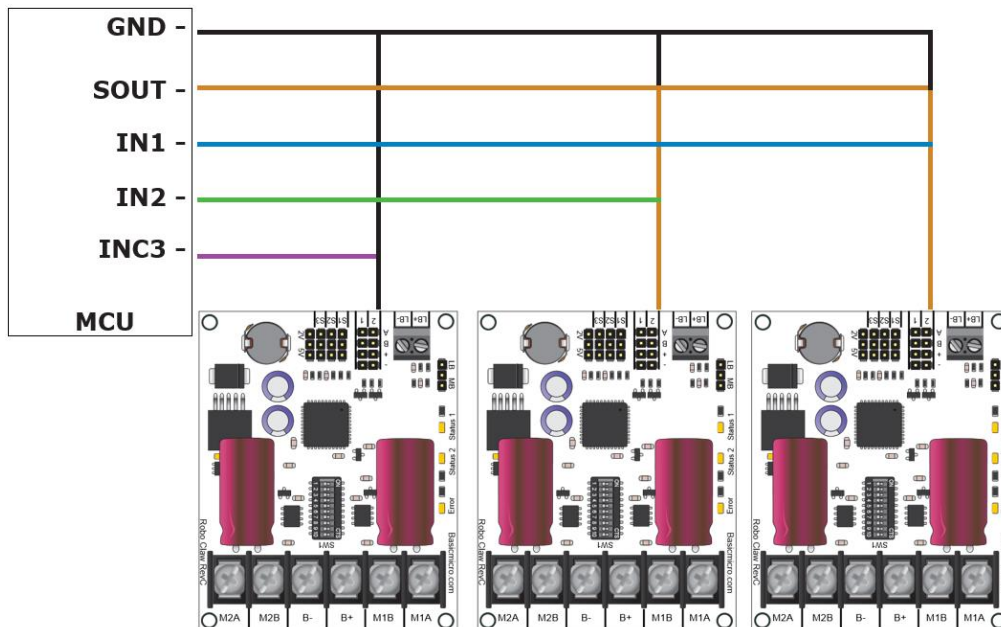
### Switch 1 - Slave Select

SW1 = ON: Turns slave select ON. Slave select is used when more than one RoboClaw is on the same serial bus. When slave select is set to ON the S2 pin becomes the select pin. Set S2 high (5V) and RoboClaw will execute the next commands. Set S2 low (0V) and RoboClaws will ignore all sent commands.



### Simple Serial Slave

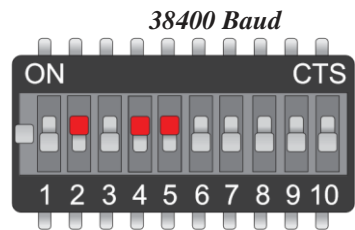
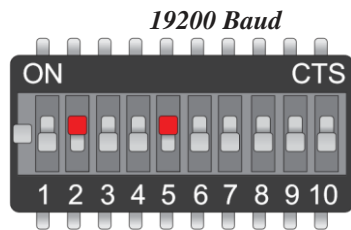
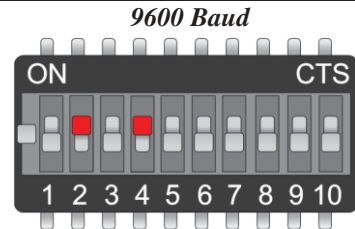
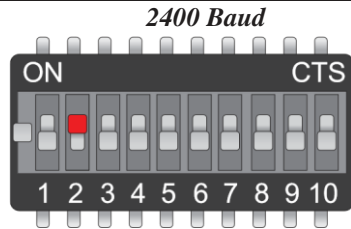
Setting up the RoboClaw for serial slave is straight forward. Make sure all RoboClaws share a common signal ground (GND) shown by the black wire. P0 (Brown line) is connected to S1 of all 3 RoboClaws which is the serial in of the RoboClaw. P1, P2 and P3 are connected to S2. Only one MCU pin is connected to each RoboClaws S2 pin. To enable RoboClaw hold S2 high otherwise any commands sent is ignored.



### Baud Rate

RoboClaw supports 4 baud rates in serial mode. The baud rate is selected by setting switch 4 and 5.

Baud Rate	SW4	SW5
2400	OFF	OFF
9600	ON	OFF
19200	OFF	ON
38400	ON	ON



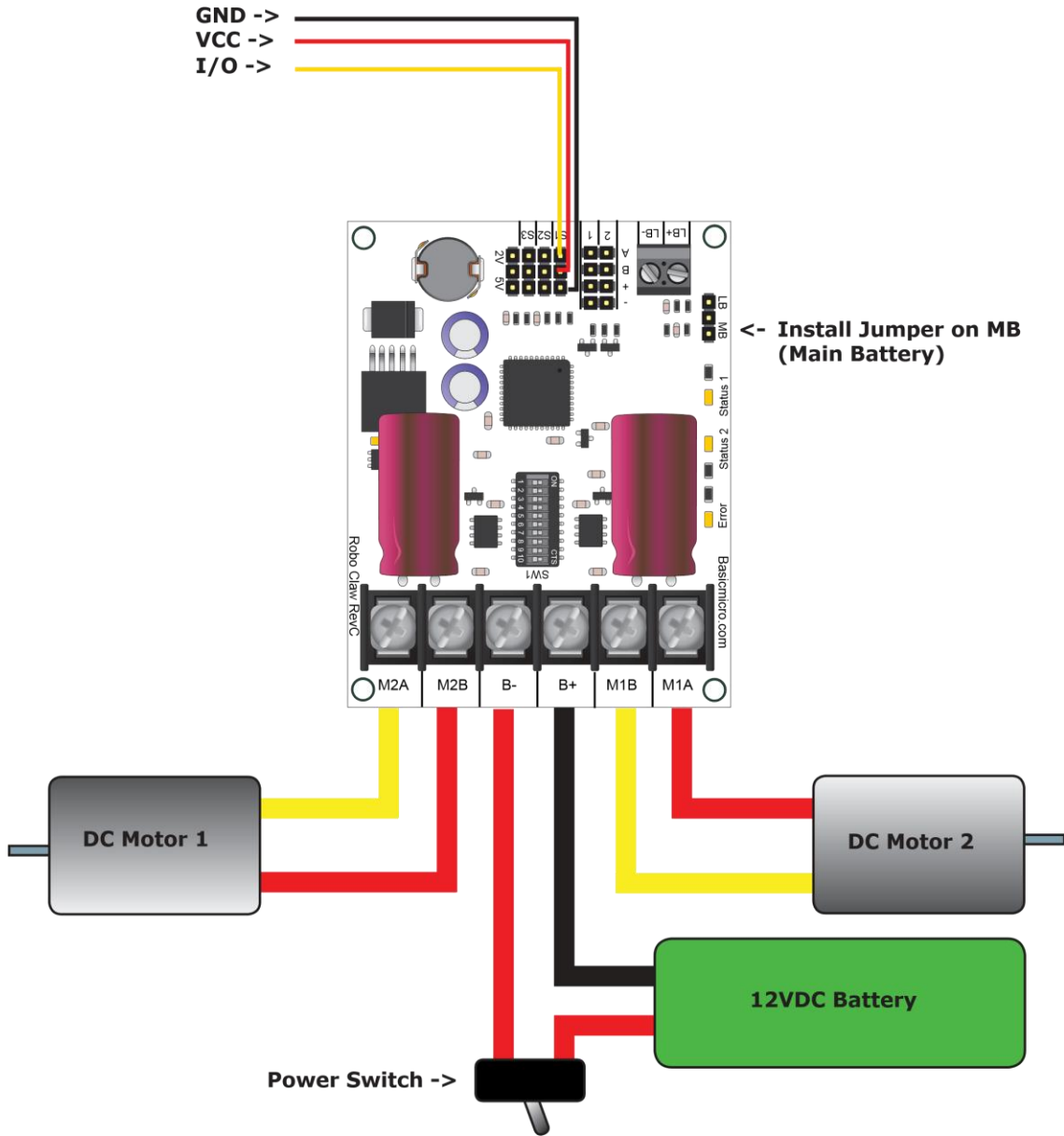
### Simple Serial Command Syntax

The RoboClaw simple serial is setup to control both motors with one byte sized command character. Since a byte can be anything from 0 to 255 the control of each motor is split. 1 to 127 controls channel 1 and 128 to 255 controls channel 2. Command character 0 will shut down both channels. Any characters in between will control speed, direction of each channel.

Character	Function
0	Shuts Down Channel 1 and 2
1	Channel 1 - Full Reverse
64	Channel 1 - Stop
127	Channel 1 - Full Forward
128	Channel 2 - Full Reverse
192	Channel 2 - Stop
255	Channel 2 - Full Forward

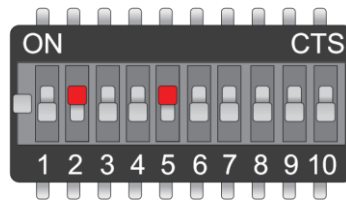
### Simple Serial Wiring Example

In simple serial mode the RoboClaw can only receive serial data. Use the below wiring diagram with the following code examples. Make sure you install the BEC jumper to 5V if powering the MCU from RoboClaw.



### Simple Serial - Arduino Example

The following example will start both channels in reverse, stop, then full speed forward. The program was written and tested with a Arduino Uno and Pin 5 connected to S1. Set switch SW2 and SW5 to ON.



```
//Basic Micro Robo Claw Simple Serial Test
//Switch settings: SW2=ON and SW5=ON //Make sure Arduino and Robo Claw share
common GND!

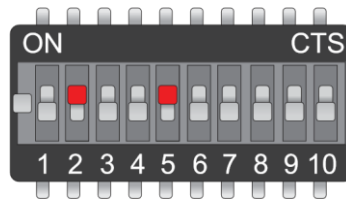
#include "BMSerial.h" BMSerial mySerial(5,6);

void setup() {
  mySerial.begin(19200);
}

void loop() { mySerial.write(1); mySerial.write(-1); delay(2000); mySerial.write(127);
mySerial.write(-127);
  delay(2000);
}
```

### Simple Serial - BasicATOM Pro Example

The following example will start both channels in reverse, stop, then full speed forward. The program was written and tested with a BasicATOM Pro and P0 connected to S1. Set switch SW2 and SW5 to ON.



```
;Basic Micro Robo Claw Simple Serial Test
;Switch settings: SW2=ON and
SW5=ON ;Make sure BAP and Robo
Claw share common GND!

Main
    Serout P15, i19200, [0] ;Full stop both channels
    Pause 500
    Serout P15, i19200, [96,224] ;Foward slowly
    Pause 3000
    Serout P15, i19200, [127,255] ;Foward
fast
    Pause 3000

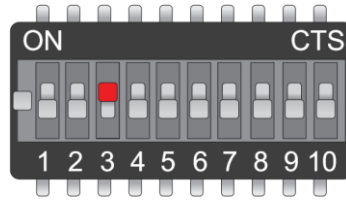
    Serout P15, i19200, [64,192] ;Full stop both channels
    Pause 500
    Serout P15, i19200, [32,160] ;Reverse slowly
    Pause 3000
    Serout P15, i19200, [1,128] ;Reverse fast
    Pause 3000
Goto Main
```

# Packet Serial



#### Mode 4 - Packet Serial

Packet serial mode set SW3 = ON and then selected address. See table below. Packet serial is used to communicate more sophisticated instructions to RoboClaw. RoboClaw can send or receive serial data in packet mode. The basic command structures consists of address byte, command byte, data bytes and a checksum. The amount of data each command will send or receive can vary. In packet mode the RoboClaw serial commands are buffered for more complex functionality.



#### Baud Rate

Packet serial supports the same baud rate modes as simple serial and uses the same RS232 8N1 format. The following table defines the available baud rates and their respective switch settings.

Baud Rate	SW4	SW5
2400	OFF	OFF
9600	ON	OFF
19200	OFF	ON
38400	ON	ON

#### Address

When using packet serial each RoboClaw must be assigned a unique address. With up to 8 addresses available you can have up to 8 RoboClaws bussed on the same RS232 port. The following table defines the addresses and their respective switch settings.

Address	SW1	SW6	SW7
128 (0x80)	OFF	OFF	OFF
129 (0x81)	OFF	ON	OFF
130 (0x82)	OFF	OFF	ON
131 (0x83)	OFF	ON	ON
132 (0x84)	ON	OFF	OFF
133 (0x85)	ON	ON	OFF
134 (0x86)	ON	OFF	ON
135 (0x87)	ON	ON	ON

### Checksum Calculation

All packet serial commands use a 7 bit checksum to prevent corrupt commands from being executed. Since the RoboClaw expects a 7bit value the 8th bit is masked. The checksum is calculated as follows:

$$\text{Address} + \text{Command} + \text{Data} = \text{Checksum}$$

To mask the 8th bit you use can a simple math expression called AND as shown below:

```
Serout P15, i19200, [128, 0, 127, (255 & 0X7F)]
```

The hexadecimal value 0X7F is used to mask the 8th bit. You can also use a binary value of 01111111 as shown below:

```
Serout P15, i19200, [128, 0, 127, (255 & %01111111)]
```

## Commands 0 - 7 Standard Commands

The following commands are the standard set of commands used with packet mode. The command syntax is the same for commands 0 to 7:

*Address, Command, ByteValue, Checksum*

### **0 - Drive Forward M1**

Drive motor 1 forward. Valid data range is 0 - 127. A value of 127 = full speed forward, 64 = about half speed forward and 0 = full stop. Example with RoboClaw address set to 128:

Serout P15, i19200, [128, 0, 127, (255 & 0X7F)] ;M1 full speed forward

### **1 - Drive Backwards M1**

Drive motor 1 backwards. Valid data range is 0 - 127. A value of 127 full speed backwards, 64 = about half speed backward and 0 = full stop. Example with RoboClaw address set to 128:

Serout P15, i19200, [128, 1, 127, (256 & 0X7F)] ;M1 full speed forward

### **2 - Set Minimum Main Voltage**

Sets main battery (B- / B+) minimum voltage level. If the battery voltages drops below the set voltage level RoboClaw will shut down. The value is cleared at start up and must set after each power up. The voltage is set in .2 volt increments. A value of 0 sets the minimum value allowed which is 6V. The valid data range is 0 - 120 (6V - 30V). The formula for calculating the voltage is: (Desired Volts - 6) x 5 = Value. Examples of valid values are 6V = 0, 8V = 10 and 11V = 25. Example with RoboClaw address set to 128:

Serout P15, i19200, [128, 2, 25, (165 & 0X7F)]

### **3 - Set Maximum Main Voltage**

Sets main battery (B- / B+) maximum voltage level. The valid data range is 0 - 154 (0V - 30V). If you are using a battery of any type you can ignore this setting. During regenerative braking a back voltage is applied to charge the battery. When using an ATX type power supply if it senses anything over 16V it will shut down. By setting the maximum voltage level, RoboClaw before exceeding it will go into hard breaking mode until the voltage drops below the maximum value set. The formula for calculating the voltage is: Desired Volts x 5.12 = Value. Examples of valid values are 12V = 62, 16V = 82 and 24V = 123. Example with RoboClaw address set to 128:

Serout P15, i19200, [128, 3, 82, (213 & 0X7F)]

### **4 - Drive Forward M2**

Drive motor 2 forward. Valid data range is 0 - 127. A value of 127 full speed forward, 64 = about half speed forward and 0 = full stop. Example with RoboClaw address set to 128:

Serout P15, i19200, [128, 4, 127, (259 & 0X7F)] ;M2 full speed forward

### **5 - Drive Backwards M2**

Drive motor 2 backwards. Valid data range is 0 - 127. A value of 127 = full speed backwards, 64 = about half speed backward and 0 = full stop. Example with RoboClaw address set to 128:

```
Serout P15, i19200, [128, 5, 127, (260 & 0X7F)] ;M2 full speed forward
```

### **6 - Drive M1 (7 Bit)**

Drive motor 1 forward and reverse. Valid data range is 0 - 127. A value of 0 = full speed reverse, 64 = stop and 127 = full speed forward. Example with RoboClaw address set to 128:

```
Serout P15, i19200, [128, 6, 96, (230 & 0X7F)] ;M1 half speed forward
```

### **7 - Drive M2 (7 Bit)**

Drive motor 2 forward and reverse. Valid data range is 0 - 127. A value of 0 = full speed reverse, 64 = stop and 127 = full speed forward. Example with RoboClaw address set to 128:

```
Serout P15, i19200, [128, 7, 32, (167 & 0X7F)] ;M2 half speed reverse
```

### Commands 8 - 13 Mix Mode Commands

The following commands are mix mode commands and used to control speed and turn. Before a command is executed valid drive and turn data is required. You only need to send both data packets once. After receiving both valid drive and turn data RoboClaw will begin to operate. At this point you only need to update turn or drive data.

#### 8 - Drive Forward

Drive forward in mix mode. Valid data range is 0 - 127. A value of 0 = full stop and 127 = full forward. Example with RoboClaw address set to 128:

```
Serout P15, i19200, [128, 8, 127, (263 & 0x7F)] ;full speed forward
```

#### 9 - Drive Backwards

Drive backwards in mix mode. Valid data range is 0 - 127. A value of 0 = full stop and 127 = full reverse. Example with RoboClaw address set to 128:

```
Serout P15, i19200, [128, 9, 127, (264 & 0x7F)] ;full speed reverse
```

#### 10 - Turn right

Turn right in mix mode. Valid data range is 0 - 127. A value of 0 = stop turn and 127 = full speed turn. Example with RoboClaw address set to 128:

```
Serout P15, i19200, [128, 10, 127, (265 & 0x7F1)] ;full speed right turn
```

#### 11 - Turn left

Turn left in mix mode. Valid data range is 0 - 127. A value of 0 = stop turn and 127 = full speed turn. Example with RoboClaw address set to 128:

```
Serout P15, i19200, [128, 11, 127, (266 & 0x7F)] ;full speed left turn
```

#### 12 - Drive Forward or Backward (7 Bit)

Drive forward or backwards. Valid data range is 0 - 127. A value of 0 = full backward, 64 = stop and 127 = full forward. Example with RoboClaw address set to 128:

```
Serout P15, i19200, [128, 12, 96, (236 & 0x7F)] ;medium speed forward
```

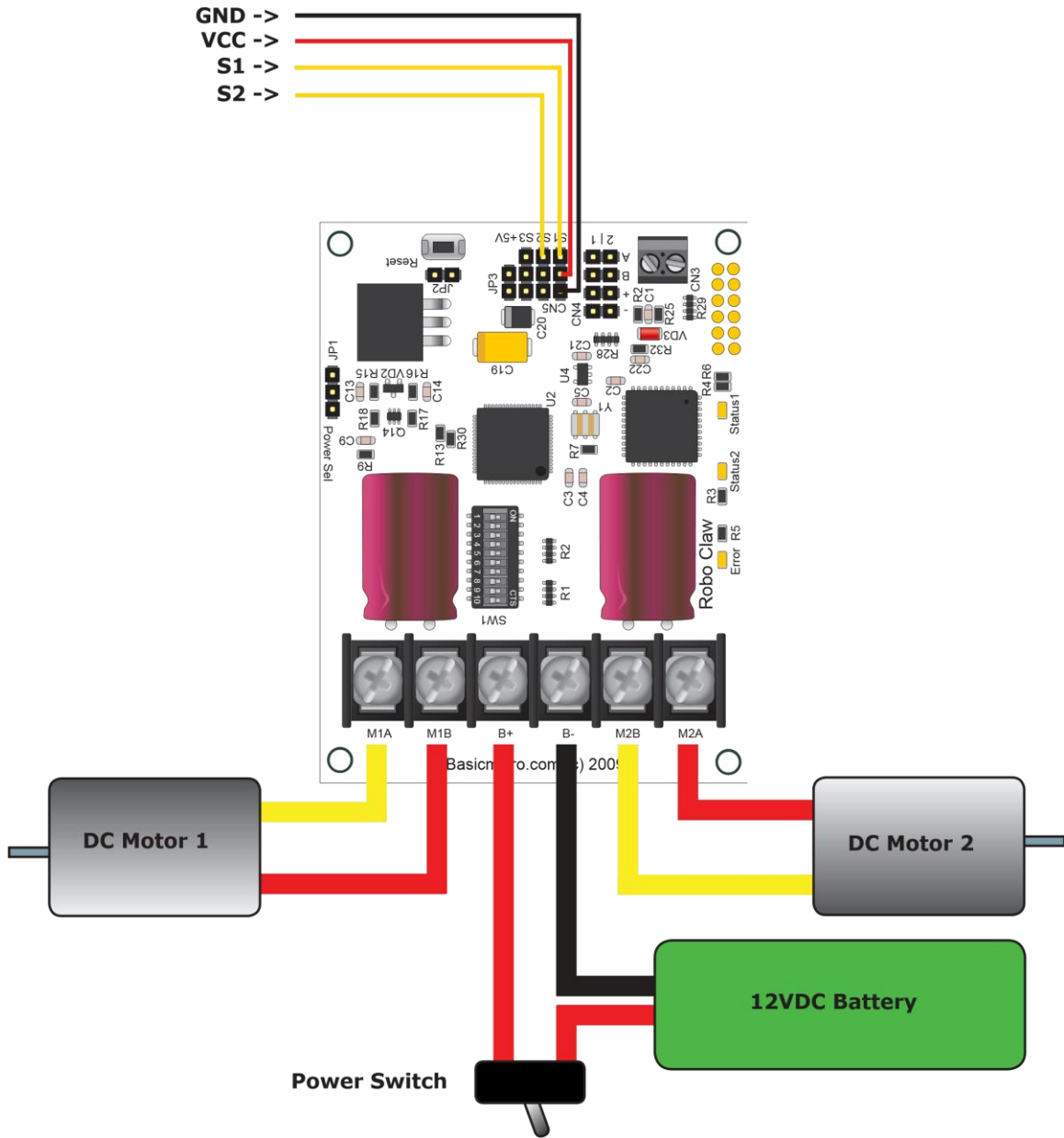
#### 13 - Turn Left or Right (7 Bit)

Turn left or right. Valid data range is 0 - 127. A value of 0 = full left, 0 = stop turn and 127 = full right. Example with RoboClaw address set to 128:

```
Serout P15, i19200, [128, 13, 0, (141 & 0x7F)] ;full speed turn left
```

### Packet Serial Wiring

In packet mode the RoboClaw can transmit and receive serial data. RoboClaw is transmitting return data a processor with a hardware serial port is required.



### Packet Serial - Arduino Example

The example will start the motor channels independently. Then start turns with mix mode commands. The program was written and tested with a Arduino Uno and P5 connected to S1. Set switch SW3 and SW5 to ON.

```
//Basic Micro Robo Claw Packet Serial Test Commands 0 to 13. //Switch settings: SW3=ON
and SW5=ON.

#include "BMSerial.h"
#include "RoboClaw.h"

#define address 0x80 RoboClaw roboclaw(5,6);

void setup() {
  roboclaw.begin(19200);
}

void loop() {
  roboclaw.ForwardM1(address,64); //Cmd 0   roboclaw.BackwardM2(address,64); //Cmd 5
  delay(2000);
  roboclaw.BackwardM1(address,64); //Cmd 1   roboclaw.ForwardM2(address,64); //Cmd 6
  delay(2000);
  roboclaw.ForwardBackwardM1(address,96);    //Cmd 6
  roboclaw.ForwardBackwardM2(address,32);    //Cmd 7  delay(2000);
  roboclaw.ForwardBackwardM1(address,32);    //Cmd 6
  roboclaw.ForwardBackwardM2(address,96);    //Cmd 7  delay(2000);

  //stop motors
  roboclaw.ForwardBackwardM1(address,0);  roboclaw.ForwardBackwardM2(address,0);
  delay(10000);

  roboclaw.ForwardMixed(address, 64); //Cmd 8  delay(2000);
  roboclaw.BackwardMixed(address, 64); //Cmd 9  delay(2000);
  roboclaw.TurnRightMixed(address, 64); //Cmd 10 delay(2000);
  roboclaw.TurnLeftMixed(address, 64);  //Cmd 11 delay(2000);
  roboclaw.ForwardBackwardMixed(address, 32); //Cmd 12 delay(2000);
  roboclaw.ForwardBackwardMixed(address, 96); //Cmd 12 delay(2000);
  roboclaw.LeftRightMixed(address, 32); //Cmd 13 delay(2000);
  roboclaw.LeftRightMixed(address, 96); //Cmd 13 delay(2000);

  //stop motors
  roboclaw.ForwardMixed(address, 0);

  delay(10000);
}
```

### Packet Serial - BasicATOM Pro Example

The example will start the motor channels independently. Then start turns with mix mode commands. The program was written and tested with a BasicATOM Pro and P15 connected to S1. Set switch SW3 and SW5 to ON.

```
;Basic Micro Robo Claw Packet Serial Test Commands 0 to 13. ;Switch settings:  
SW3=ON and SW5=ON.
```

```
Main
```

```
    Pause 2000
```

```
        Serout P15, i19200, [128, 0, 127, (255 & 0x7F)];M1 full speed forward
```

```
Serout P15, i19200, [128, 4, 127, (259 & 0x7F)];M2 full speed forward
```

```
    Pause 1000
```

```
        Serout P15, i19200, [128, 0, 0, (128 & 0x7F)];M1 stop          Serout P15,  
i19200, [128, 4, 0, (132 & 0x7F)];M2 stop
```

```
    Pause 1000
```

```
Serout P15, i19200, [128, 1, 127, (256 & 0x7F)];M1 full speed backwards Serout P15,  
i19200, [128, 5, 127, (260 & 0x7F)];M1 full speed backwards
```

```
    Pause 1000
```

```
        Serout P15, i19200, [128, 0, 0, (128 & 0x7F)];M1 stop          Serout P15,  
i19200, [128, 4, 0, (132 & 0x7F)];M2 stop
```

```
    Pause 1000
```

```
        Serout P15, i19200, [128, 10, 127, (265 & 0x7F)];Mix mode right full speed
```

```
    Pause 1000
```

```
Serout P15, i19200, [128, 10, 0, (138 & 0x7F)];Mix mode stop
```

```
    Pause 1000
```

```
        Serout P15, i19200, [128, 11, 127, (266 & 0x7F)];Mix mode left full speed
```

```
    Pause 1000
```

```
Serout P15, i19200, [128, 11, 0, (139 & 0x7F)];Mix mode stop Goto Main
```



# Battery and Version Information

## 21 - Read Firmware Version

Read RoboClaw firmware version. Returns up to 32 bytes and is terminated by a null character. Command syntax:

```
Sent: [Address, CMD]
Received: ["RoboClaw 10.2A v1.3.9, Checksum]
```

The command will return up to 32 bytes. The return string includes the product name and firmware version. The return string is terminated with a null (0) character. This is done so the version information can be read from a standard PC terminal window.

```
hserout [128, 21] ;read firmware version
hserin [Str VersionByte\32\0, Checksum]
```

## 24 - Read Main Battery Voltage Level

Read the main battery voltage level connected to B+ and B- terminals. The voltage is returned in 10ths of a volt. Command syntax:

```
Sent: [Address, CMD]
Received: [Value.Byte1, Value.Byte0, Checksum]
```

The command will return 3 bytes. Byte 1 and 2 make up a word variable which is received MSB first and is 10th of a volt. A returned value of 300 would equal 30V. Byte 3 is the checksum. It is calculated the same way as sending a command and can be used to validate the data. The following example will read the main battery voltage with RoboClaw address set to 128.

```
hserout [128, 24] ;read main battery voltage
hserin [Value.Byte1, Value.Byte0, Checksum]
```

## 25 - Read Logic Battery Voltage Level

Read a logic battery voltage level connected to LB+ and LB- terminals. The voltage is returned in 10ths of a volt. Command syntax:

```
Sent: [Address, CMD]
Received: [Value.Byte1, Value.Byte0, Checksum]
```

The command will return 3 bytes. Byte 1 and 2 make up a word variable which is received MSB first and is 10th of a volt. A returned value of 50 would equal 5V. Byte 3 is the checksum. It is calculated the same way as sending a command and can be used to validate the data. The following example will read the main battery voltage with RoboClaw address set to 128.

```
hserout [128, 25] ;read logic battery voltage
hserin [Value.Byte1, Value.Byte0, Checksum]
```

### **26 – Set Minimum Logic Voltage Level**

Sets logic input (LB- / LB+) minimum voltage level. If the battery voltages drops below the set voltage level RoboClaw will shut down. The value is cleared at start up and must set after each power up. The voltage is set in .2 volt increments. A value of 0 sets the minimum value allowed which is 3V. The valid data range is 0 - 120 (6V - 28V). The formula for calculating the voltage is: (Desired Volts - 6) x 5 = Value. Examples of valid values are 3V = 0, 8V = 10 and 11V = 25. RoboClaw example with address set to 128:

```
hserout [128, 26, 0, (154 & 0X7F)]
```

### **27 - Set Maximum Logic Voltage Level**

Sets logic input (LB- / LB+) maximum voltage level. The valid data range is 0 - 144 (0V - 28V). By setting the maximum voltage level RoboClaw will go into shut down and requires a hard reset to recovers. The formula for calculating the voltage is: Desired Volts x 5.12 = Value. Examples of valid values are 12V = 62, 16V = 82 and 24V = 123. RoboClaw example with address set to 128: hserout [128, 27, 82, (213 & 0X7F)]

### **Main Battery Voltage Levels**

The main battery levels are set in a similar way as the logic battery. See command 2 and 3 for details.

# Quadrature Decoding

### **Quadrature Decoding**

Handling the quadrature encoders is done using packet serial. All the switch settings still apply in to enabling packet serial and setting the desired baud rates. See Mode - Packet Serial. The following commands deal specifically with the dual quadrature decoders built into RoboClaw.

### **Checksum Calculation**

All packet serial commands use a 7 bit checksum to prevent corrupt commands from being executed. Since the RoboClaw expects a 7bit value the 8th bit is masked. The checksum is calculated as follows:

$$\text{Address} + \text{Command} + \text{Data} = \text{Checksum}$$

To mask the 8th bit you use can a simple math expression called AND as shown below:

```
Serout P15, i19200, [128, 0, 127, (255 & 0X7F)]
```

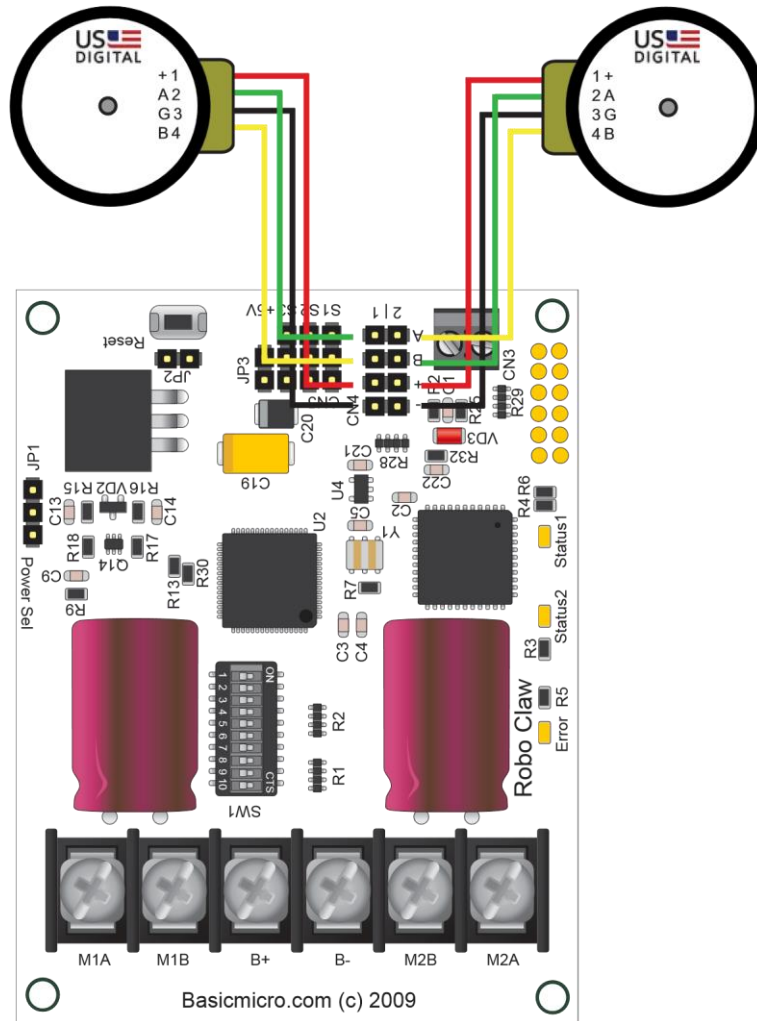
The hexadecimal value 0X7F is used to mask the 8th bit. You can also use a binary value of 01111111 as shown below:

```
Serout P15, i19200, [128, 0, 127, (255 & %01111111)]
```

## Quadrature Encoder Wiring

RoboClaw can read two quadrature encoders. The encoders are connected to RoboClaw using CN4. Both GND and 5 volts are present on the header to power the encoders.

In a two motor robot configuration one motor will spin clock wise (CW) while the other motor will spin counter clock wise (CCW). The A and B inputs for one of the two encoders must be reversed as shown. If either encoder is connected wrong one will count up and the other down this will cause commands like mix drive forward to not work properly.



### Commands 16 - 20 Reading Quadrature Encoders

The following commands are used in dealing with the quadrature decoding counter registers. The quadrature decoder is a simple counter that counts the incoming pulses, tracks the direction and speed of each pulse. There are two registers one each for M1 and M2. (Note: A microcontroller with a hardware UART is recommended for use with packet serial modes).

Command	Description
16	Read Quadrature Encoder Register for M1.
17	Read Quadrature Encoder Register for M2.
18	Read M1 Speed in Pulses Per Second.
19	Read M2 Speed in Pulses Per Second.
20	Resets Quadrature Encoder Registers for M1 and M2.

#### 16 - Read Quadrature Encoder Register M1

Read decoder M1 counter. Since CMD 16 is a read command it does not require a checksum. However a checksum value will be returned from RoboClaw and can be used to validate the data. Command syntax:

```
Sent: [Address, CMD]
Received: [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2,
Checksum]
```

The command will return 6 bytes. Byte 1,2,3 and 4 make up a long variable which is received MSB first and represents the current count which can be any value from 0 - 4,294,967,295. Each pulse from the quadrature encoder will increment or decrement the counter depending on the direction of rotation.

Byte 5 is the status byte for M1 decoder. It tracks counter underflow, direction, overflow and if the encoder is operational. The byte value represents:

- Bit0 - Counter Underflow (1= Underflow Occurred, Clear After Reading)
- Bit1 - Direction (0 = Forward, 1 = Backwards)
- Bit2 - Counter Overflow (1= Overflow Occurred, Clear After Reading)
- Bit3 - Reserved
- Bit4 - Reserved
- Bit5 - Reserved
- Bit6 - Reserved
- Bit7 - Reserved

Byte 6 is the checksum. It is calculated the same way as sending a command. It can be used to validate the resulting data. The following example will read M1 counter register, status byte and checksum value with RoboClaw address set to 128.

```
hserout [128, 16] ;read command for M1 encoder
hserin [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]
```

### 17 - Read Quadrature Encoder Register M2

Read decoder M2 counter. Since CMD 16 is a read command it does not require a checksum. However a checksum value will be returned from RoboClaw and can be used to validate the data. Command syntax:

Sent: [Address, CMD]

Received: [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]

The command will return 6 bytes. Byte 1,2,3 and 4 make up a long variable which is received MSB first and represents the current count which can be any value from 0 - 4,294,967,295. Each pulse from the quadrature encoder will increment or decrement the counter depending on the direction of rotation.

Byte 5 is the status byte for M1 decoder. It tracks counter underflow, direction, overflow and if the encoder is operational. The byte value represents:

Bit0 - Counter Underflow (1= Underflow Occurred, Clear After Reading)

Bit1 - Direction (0 = Forward, 1 = Backwards)

Bit2 - Counter Overflow (1= Overflow Occurred, Clear After Reading)

Bit3 - Reserved

Bit4 - Reserved

Bit5 - Reserved

Bit6 - Reserved

Bit7 - Reserved

Byte 6 is the checksum. It is calculated the same way as sending a command. It can be used to validate the resulting data. The following example will read M1 counter register, status byte and checksum value with RoboClaw address set to 128.

```
hserout [128, 17] ;read command for M2 encoder
```

```
hserin [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]
```

### 18 - Read Speed M1

Read M1 counter speed. Returned value is in pulses per second. RoboClaw keeps track of how many pulses received per second for both decoder channels. Since CMD 18 is a read command it does not require a checksum to be sent. However a checksum value will be returned from RoboClaw and can be used to validate the data. Command syntax:

Sent: [Address, CMD]

Received: [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]

The command will return 6 bytes. Byte 1,2,3 and 4 make up a long variable which is received MSB first and is the current ticks per second which can be any value from 0 - 4,294,967,295. Byte 5 is the direction (0 – forward, 1 - backward). Byte 6 is the checksum. It is calculated the same way as sending a command and can be used to validate the data. The following example will read M1 pulse per second and direction with RoboClaw address set to 128.

```
hserout [128, 18] ;read command for M1 encoder
```

```
hserin [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]
```



### 19 - Read Speed M2

Read M2 counter speed. Returned value is in pulses per second. RoboClaw keeps track of how many pulses received per second for both decoder channels. Since CMD 19 is a read command it does not require a checksum to be sent. However a checksum value will be returned from RoboClaw and can be used to validate the data. Command syntax:

Sent: [Address, CMD]

Received: [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]

The command will return 6 bytes. Byte 1,2,3 and 4 make up a long variable which is received MSB first and is the current ticks per second which can be any value from 0 - 4,294,967,295. Byte 5 is the direction (0 – forward, 1 - backward). Byte 6 is the checksum. It is calculated the same way as sending a command and can be used to validate the data. The following example will read M2 pulse per second and direction with RoboClaw address set to 128.

hserout [128, 19] ;read command for M2 encoder

hserin [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]

### 20 - Reset Quadrature Encoder Counters

Will reset both quadrature decoder counters to zero. Since CMD 20 is a write command a checksum value is required. Command syntax and example:

hserout [128, 20, (148 & %01111111)]; resets encoder registers

### Commands 28 - 48 Motor Control by Quadrature Encoders

The following commands are used to control motor speeds, acceleration and distance using the quadrature encoders. All speeds are given in quad pulses per second (QPPS) unless otherwise stated. Quadrature encoders of different types and manufactures can be used. However many have different resolutions and maximum speeds at which they operate. So each quadrature encoder will produce a different range of pulses per second.

Command	Description
28	Set PID Constants for M1.
29	Set PID Constants for M2.
30	Read Current M1 Speed Resolution 125th of a Second.
31	Read Current M2 Speed Resolution 125th of a Second.
32	Drive M1 With Signed Duty Cycle.
33	Drive M2 With Signed Duty Cycle.
34	Mix Mode Drive M1 / M2 With Signed Duty Cycle.
35	Drive M1 With Signed Speed.
36	Drive M2 With Signed Speed.
37	Mix Mode Drive M1 / M2 With Signed Speed.
38	Drive M1 With Signed Speed And Acceleration.
39	Drive M2 With Signed Speed And Acceleration.
40	Mix Mode Drive M1 / M2 With Speed And Acceleration.
41	Drive M1 With Signed Speed And Distance. Buffered.
42	Drive M2 With Signed Speed And Distance. Buffered.
43	Mix Mode Drive M1 / M2 With Speed And Distance. Buffered.
44	Drive M1 With Signed Speed, Acceleration and Distance. Buffered.
45	Drive M2 With Signed Speed, Acceleration and Distance. Buffered.
46	Mix Mode Drive M1 / M2 With Speed, Acceleration And Distance. Buffered.
47	Read Buffer Length.
48	Set PWM Resolution.

## 28 Set PID Constants M1

Several motor and quadrature combinations can be used with RoboClaw. In some cases the default PID values will need to be tuned for the systems being driven. This gives greater flexibility in what motor and encoder combinations can be used. The RoboClaw PID system consists of four constants starting with QPPS, P = Proportional, I= Integral and D= Derivative. The default values are:

QPPS = 44000  
P = 0x00010000  
I = 0x00008000  
D = 0x00004000

QPPS is the speed of the encoder when the motor is at 100% power. P, I, D are the default values used after a reset. Command syntax:

Sent: [Address, CMD, D(4 bytes), P(4 bytes), I(4 bytes), QPPS(4 byte), Checksum]

Each value is made up of 4 bytes for a long. To write the registers a checksum value is used. This prevents an accidental write.

## 29 - Set PID Constants M2

Several motor and quadrature combinations can be used with RoboClaw. In some cases the default PID values will need to be tuned for the systems being driven. This gives greater flexibility in what motor and encoder combinations can be used. The RoboClaw PID system consists of four constants starting with QPPS, P = Proportional, I= Integral and D= Derivative. The default values are:

QPPS = 44000  
P = 0x00010000  
I = 0x00008000  
D = 0x00004000

QPPS is the speed of the encoder when the motor is at 100% power. P, I, D are the default values used after a reset. Command syntax:

Sent: [Address, CMD, D(4 bytes), P(4 bytes), I(4 bytes), QPPS(4 byte), Checksum]

Each value is made up of 4 bytes for a long. To write the registers a checksum value is used. This prevents an accidental write.

## 30 - Read Current Speed M1

Read the current pulse per 125th of a second. This is a high resolution version of command 18 and 19. Command 30 can be used to make an independent PID routine. The resolution of the command is required to create a PID routine using any microcontroller or PC used to drive RoboClaw. The command syntax:

Sent: [Address, CMD]  
Received: [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]

The command will return 5 bytes, MSB sent first for a long. The first 4 bytes are a 32 byte value (long) that represent the speed. The 5th byte (Value2) is direction (0 – forward, 1 - backward). A checksum is returned in order to validate the data returned.

### **31 - Read Current Speed M2**

Read the current pulse per 125th of a second. This is a high resolution version of command 18 and 19. Command 31 can be used to make a independent PID routine. The resolution of the command is required to create a PID routine using any microcontroller or PC used to drive RoboClaw. The command syntax:

Sent: [Address, CMD]

Received: [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]

The command will return 5 bytes, MSB sent first for a long. The first 4 bytes are a 32 byte value (long) that represent the speed. The 5th byte (Value2) is direction (0 – forward, 1 - backward). A checksum is returned in order to validate the data returned.

### **32 - Drive M1 With Signed Duty Cycle**

Drive M1 using a duty cycle value. The default PWM is 8bit resolution. The default value can be changed see CMD 48. The duty cycle is used to control the speed of the motor without a quadrature encoder. A value used to drive one motor at 50% will be differ from one motor to the next. The command syntax:

Sent: [Address, CMD, Duty(2 Bytes), Checksum]

The duty value is signed and the default range is 8bits. The default PWM resolution can be changed for more range. To change the resolution see command 48.

### **33 - Drive M2 With Signed Duty Cycle**

Drive M2 using a duty cycle value. The default PWM is 8bit resolution. The default value can be changed see CMD 48. The duty cycle is used to control the speed of the motor without a quadrature encoder. A value used to drive one motor at 50% will be differ from one motor to the next. The command syntax:

Sent: [Address, CMD, Duty(2 Bytes), Checksum]

The duty value is signed and the default range is 8bits. The default PWM resolution can be changed for more range. To change the resolution see command 48.

### **34 - Mix Mode Drive M1 / M2 With Signed Duty Cycle**

Drive both M1 and M2 using a duty cycle value. The default PWM is 8bit resolution. The default value can be changed see CMD 48. The duty cycle is used to control the speed of the motor without a quadrature encoder. A value used to drive one motor at 50% will be differ from one motor to the next. The command syntax:

Sent: [Address, CMD, DutyM1(2 Bytes), DutyM2(2 Bytes), Checksum]

The duty value is signed and the default range is 8bits. The default PWM resolution can be changed for more range. To change the resolution see command 48.

### **35 - Drive M1 With Signed Speed**

Drive M1 using a speed value. The sign indicates which direction the motor will turn. This command is used to drive the motor by quad pulses per second. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once

a value is sent the motor will begin to accelerate as fast as possible until the defined rate is reached. The command syntax:

Sent: [Address, CMD, Qspeed(4 Bytes), Checksum]

4 Bytes (long) are used to express the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses.

### **36 - Drive M2 With Signed Speed**

Drive M2 with a speed value. The sign indicates which direction the motor will turn. This command is used to drive the motor by quad pulses per second. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once a value is sent, the motor will begin to accelerate as fast as possible until the rate defined is reached. The command syntax:

Sent: [Address, CMD, Qspeed(4 Bytes), Checksum]

4 Bytes (long) are used to expressed the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses.

### **37 - Mix Mode Drive M1 / M2 With Signed Speed**

Drive M1 and M2 in the same command using a signed speed value. The sign indicates which direction the motor will turn. This command is used to drive both motors by quad pulses per second. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once a value is sent the motor will begin to accelerate as fast as possible until the rate defined is reached. The command syntax:

Sent: [Address, CMD, QspeedM1(4 Bytes), QspeedM2(4 Bytes), Checksum]

4 Bytes (long) are used to express the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses.

### **38 - Drive M1 With Signed Speed And Acceleration**

Drive M1 with a signed speed and acceleration value. The sign indicates which direction the motor will run. The acceleration values are not signed. This command is used to drive the motor by quad pulses per second and using an acceleration value for ramping. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once a value is sent the motor will begin to accelerate incrementally until the rate defined is reached. The command syntax:

Sent: [Address, CMD, Accel(4 Bytes), Qspeed(4 Bytes), Checksum]

4 Bytes (long) are used to express the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses. The acceleration is measured in speed per second. An acceleration value of 12,000 QPPS with a speed of 12,000 QPPS would accelerate a motor from 0 to 12,000 QPPS in 1 second. Another example would be an acceleration value of 24,000 QPPS and a speed value of 12,000 QPPS would accelerate the motor to 12,000 QPPS in 0.5 seconds.

### **39 - Drive M2 With Signed Speed And Acceleration**

Drive M2 with a signed speed and acceleration value. The sign indicates which direction the motor will run. The acceleration value is not signed. This command is used to drive the motor by quad pulses per second and using an acceleration value for ramping. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once a value is sent the motor will begin to accelerate incrementally until the rate defined is reached. The command syntax:

Sent: [Address, CMD, Accel(4 Bytes), Qspeed(4 Bytes), Checksum]

4 Bytes (long) are used to express the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses. The acceleration is measured in speed per second. An acceleration value of 12,000 QPPS with a speed of 12,000 QPPS would accelerate a motor from 0 to 12,000 QPPS in 1 second. Another example would be an acceleration value of 24,000 QPPS and a speed value of 12,000 QPPS would accelerate the motor to 12,000 QPPS in 0.5 seconds.

### **40 - Mix Mode Drive M1 / M2 With Speed And Acceleration**

Drive M1 and M2 in the same command using one value for acceleration and two signed speed values for each motor. The sign indicates which direction the motor will run. The acceleration value is not signed. The motors are sync during acceleration. This command is used to drive the motor by quad pulses per second and using an acceleration value for ramping. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once a value is sent the motor will begin to accelerate incrementally until the rate defined is reached. The command syntax:

Sent: [Address, CMD, Accel(4 Bytes), QspeedM1(4 Bytes), QspeedM2(4 Bytes), Checksum]

4 Bytes (long) are used to express the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses. The acceleration is measured in speed per second. An acceleration value of 12,000 QPPS with a speed of 12,000 QPPS would accelerate a motor from 0 to 12,000 QPPS in 1 second. Another example would be an acceleration value of 24,000 QPPS and a speed value of 12,000 QPPS would accelerate the motor to 12,000 QPPS in 0.5 seconds.

### **41 - Buffered M1 Drive With Signed Speed And Distance**

Drive M1 with a signed speed and distance value. The sign indicates which direction the motor will run. The distance value is not signed. This command is buffered. This command is used to control the top speed and total distance traveled by the motor. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second. The command syntax:

Sent: [Address, CMD, QSpeed(4 Bytes), Distance(4 Bytes), Buffer(1 Byte), Checksum]

4 Bytes(long) are used to express the pulses per second. The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

#### **42 - Buffered M2 Drive With Signed Speed And Distance**

Drive M2 with a speed and distance value. The sign indicates which direction the motor will run. The distance value is not signed. This command is buffered. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second. The command syntax:

Sent: [Address, CMD, QSpeed(4 Bytes), Distance(4 Bytes), Buffer(1 Byte), Checksum]

4 Bytes(long) are used to express the pulses per second. The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

#### **43 - Buffered Mix Mode Drive M1 / M2 With Signed Speed And Distance**

Drive M1 and M2 with a speed and distance value. The sign indicates which direction the motor will run. The distance value is not signed. This command is buffered. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second. The command syntax:

Sent: [Address, CMD, QSpeedM1(4 Bytes), DistanceM1(4 Bytes),  
QSpeedM2(4 Bytes), DistanceM2(4 Bytes), Buffer(1 Byte), Checksum]

4 Bytes(long) are used to express the pulses per second. The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

#### **44 - Buffered M1 Drive With Signed Speed, Accel And Distance**

Drive M1 with a speed, acceleration and distance value. The sign indicates which direction the motor will run. The acceleration and distance values are not signed. This command is used to control the motors top speed, total distanced traveled and at what incremental acceleration value to use until the top speed is reached. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second. The command syntax:

Sent: [Address, CMD, Accel(4 bytes), QSpeed(4 Bytes), Distance(4 Bytes), Buffer(1 Byte),  
Checksum]

4 Bytes(long) are used to express the pulses per second. The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

#### **45 - Buffered M2 Drive With Signed Speed, Accel And Distance**

Drive M2 with a speed, acceleration and distance value. The sign indicates which direction the motor will run. The acceleration and distance values are not signed. This command is used to control the motors top speed, total distanced traveled and at what incremental acceleration value to use until the top speed is reached. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second. The command syntax:

Sent: [Address, CMD, Accel(4 bytes), QSpeed(4 Bytes), Distance(4 Bytes), Buffer(1 Byte), Checksum]

4 Bytes(long) are used to express the pulses per second. The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

#### **46 - Buffered Mix Mode Drive M1 / M2 With Signed Speed, Accel And Distance**

Drive M1 and M2 with a speed, acceleration and distance value. The sign indicates which direction the motor will run. The acceleration and distance values are not signed. This command is used to control both motors top speed, total distanced traveled and at what incremental acceleration value to use until the top speed is reached. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second. The command syntax:

Sent: [Address, CMD, Accel(4 Bytes), QSpeedM1(4 Bytes), DistanceM1(4 Bytes), QSpeedM2(4 bytes), DistanceM2(4 Bytes), Buffer(1 Byte), Checksum]

4 Bytes(long) are used to express the pulses per second. The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

#### **47 - Read Buffer Length**

Read both motor M1 and M2 buffer lengths. This command can be used to determine how many commands are waiting to execute.

Sent: [Address, CMD]

Received: [BufferM1(1 Bytes), BufferM2(1 Bytes), Checksum]

The return values represent how many commands per buffer are waiting to be executed. The maximum buffer size per motor is 31 commands.



## Reading Quadrature Encoder - Arduino Example

The example was tested with an Arduino Uno. RoboClaw was connected as shown in both packet serial wiring and quadrature encoder wiring diagrams.

The example will read the speed, total ticks and direction of each encoder. Connect to the program using a terminal window set to 38400 baud. The program will display the values of each encoders current count along with each encoder status bit in binary and the direction bit. As the encoder is turned it will update the screen.

```
//Basic Micro Robo Claw Packet Serial Mode.
//Switch settings: SW3=ON, SW4=ON, SW5=ON

#include "BMSerial.h"
#include "RoboClaw.h"

#define address 0x80

#define Kp 0x00010000
#define Ki 0x00008000
#define Kd 0x00004000
#define qpps 44000

BMSerial terminal(0,1); RoboClaw roboclaw(5,6);

void setup() { terminal.begin(38400); roboclaw.begin(38400);

  roboclaw.SetM1Constants(address,Kd,Kp,Ki,qpps);
  roboclaw.SetM2Constants(address,Kd,Kp,Ki,qpps);
}

void loop() { uint8_t status; bool valid;

  uint32_t enc1= roboclaw.ReadEncM1(address, &status, &valid); if(valid){
    terminal.print("Encoder1:"); terminal.print(enc1,HEX); terminal.print(" ");
    terminal.print(status,HEX); terminal.print(" ");
  }
  uint32_t enc2 = roboclaw.ReadEncM2(address, &status, &valid); if(valid){
    terminal.print("Encoder2:"); terminal.print(enc2,HEX); terminal.print(" ");
    terminal.print(status,HEX); terminal.print(" ");
  }
  uint32_t speed1 = roboclaw.ReadSpeedM1(address, &status, &valid); if(valid){
    terminal.print("Speed1:"); terminal.print(speed1,HEX); terminal.print(" ");
  }
  uint32_t speed2 = roboclaw.ReadSpeedM2(address, &status, &valid); if(valid){
    terminal.print("Speed2:"); terminal.print(speed2,HEX); terminal.print(" ");
  }
  terminal.println();

  delay(100);
}
```

### Speed Controlled by Quadrature Encoders - Arduino Example

The following example was written using an BasicATOM Pro. RoboClaw was connected as shown in both packet serial wiring and quadrature encoder wiring diagrams.

The example will command a 4wheel robot to move forward, backward, right turn and left turn slowly. You can change the speed by adjusting the value of Speed and Speed2 variables.

```
//Basic Micro Robo Claw Packet Serial Mode.
//Switch settings: SW3=ON, SW4=ON, SW5=ON

#include "BMSerial.h"
#include "RoboClaw.h"

#define address 0x80

#define Kp 0x00010000
#define Ki 0x00008000
#define Kd 0x00004000
#define qpps 44000

BMSerial terminal(0,1); RoboClaw roboclaw(5,6);

void setup() { terminal.begin(38400); roboclaw.begin(38400);

  roboclaw.SetM1Constants(address,Kd,Kp,Ki,qpps);
  roboclaw.SetM2Constants(address,Kd,Kp,Ki,qpps);
}

void displayspeed(void)
{
  uint8_t status; bool valid;

  uint32_t enc1= roboclaw.ReadEncM1(address, &status, &valid); if(valid){
  terminal.print("Encoder1:"); terminal.print(enc1,DEC); terminal.print(" ");
  terminal.print(status,HEX); terminal.print(" ");
  }
  uint32_t enc2 = roboclaw.ReadEncM2(address, &status, &valid); if(valid){
  terminal.print("Encoder2:"); terminal.print(enc2,DEC); terminal.print(" ");
  terminal.print(status,HEX); terminal.print(" ");
  }
}
```

```

    uint32_t speed1 = roboclaw.ReadSpeedM1(address, &status,
&valid); if(valid){ terminal.print("Speed1:");
terminal.print(speed1,DEC); terminal.print(" ");
}
    uint32_t speed2 = roboclaw.ReadSpeedM2(address, &status,
&valid); if(valid){ terminal.print("Speed2:");
terminal.print(speed2,DEC); terminal.print(" ");
}
    terminal.println();
}

void loop() {

roboclaw.SpeedAccelDistanceM1(address,12000,12000,48000);
uint8_t depth1,depth2; do{
    displayspeed();
    roboclaw.ReadBuffers(address,depth1,depth2);
}while(depth1);
roboclaw.SpeedAccelDistanceM1(address,12000,-
12000,48000); do{
    displayspeed();
    roboclaw.ReadBuffers(address,depth1,depth2);
}while(depth1);

}

```

### Reading Quadrature Encoder - BasicATOM Pro Example

The example was tested with a BasicATOM Pro. RoboClaw was connected as shown in both packet serial wiring and quadrature encoder wiring diagrams.

The example will read the speed, total ticks and direction of each encoder. Connect to the program using the Basic Micro Studio terminal window set to 38400 baud. The program will display the values of each encoders current count along with each encoder status bit in binary and the direction bit. As the encoder is turned it will update the screen.

```
Encoder1 Var Long
Encoder2 Var Long
Status Var Byte
CRC Var Byte
ENABLEHSERIAL ;used on AtomPro24 and AtomPro28. AtomPro40 and ARC-32 use EnableHSerial2
SetHSerial H38400,H8DATABITS,HNOPARITY,H1STOPBITS
Pause 250
Hserout [128, 20, (148 & 0x7F)]; Resets encoder registers
Main
    Pause 100
ReadEncoderM1
    Hserout [128, 16]
        Hserin [Encoder1.byte3, Encoder1.Byte2, Encoder1.Byte1, Encoder1.Byte0, Status, crc]
        Serout s_out, i38400, [0,"Encoder1: ", SDEC Encoder1, 13, "Status Byte: ", BIN Status]
ReadSpeedM1
    Hserout [128, 18]
        Hserin [Encoder1.byte3, Encoder1.Byte2, Encoder1.Byte1, Encoder1.Byte0, Status, crc]
        Serout s_out, i38400, [13, 13, "Speed: ", DEC Encoder1, 13, "Direction: ", DEC Status]
ReadEncoderM2
    Hserout [128, 17]
        Hserin [Encoder2.byte3, Encoder2.Byte2, Encoder2.Byte1, Encoder2.Byte0, Status, crc]
        Serout s_out, i38400, [13, 13, "Encoder2: ", SDEC Encoder2, 13, "Status Byte: ", BIN Status]
ReadSpeedM2
    Hserout [128, 19]
        Hserin [Encoder2.byte3, Encoder2.Byte2, Encoder2.Byte1, Encoder2.Byte0, Status, crc]
        Serout s_out, i38400, [13, 13, "Speed: ", DEC Encoder2, 13, "Direction: ", DEC Status]
Goto Main
```

## Speed Controlled by Quadrature Encoders - BasicATOM Pro Example

The following example was written using an BasicATOM Pro. RoboClaw was connected as shown in both packet serial wiring and quadrature encoder wiring diagrams.

The example will command a 4wheel robot to move forward, backward, right turn and left turn slowly. You can change the speed by adjusting the value of Speed and Speed2 variables.

```
CMD var byte Speed var long Speed2 var long CRC var byte
Address con 128

ENABLEHSERIAL ;used on AtomPro24 and AtomPro28. AtomPro40 and ARC-32 use EnableHSerial2 SetHSerial
H38400,H8DATABITS,HNOPARITY,H1STOPBITS

Mixed_Forward
  CMD=37
  Speed=12000
  Speed2=12000
  CRC = (address + cmd + speed.byte3 + speed.byte2 + speed.byte1 + speed.byte0 + |
        speed2.byte3 + speed2.byte2 + speed2.byte1 + speed2.byte0)&0x7F
  [address,cmd,speed.byte3,speed.byte2,speed.byte1,speed.byte0, | hserout
  speed2.byte3,speed2.byte2,speed2.byte1,speed2.byte0,crc] pause 4000

Mixed_Backward
  CMD=37
  Speed=-12000
  Speed2=-12000
  CRC = (address + cmd + speed.byte3 + speed.byte2 + speed.byte1 + speed.byte0 + |
        speed2.byte3 + speed2.byte2 + speed2.byte1 + speed2.byte0)&0x7F
  [address,cmd,speed.byte3,speed.byte2,speed.byte1,speed.byte0, | hserout
  speed2.byte3,speed2.byte2,speed2.byte1,speed2.byte0,crc] pause 4000

Mixed_Left
  CMD=37
  Speed=-12000
  Speed2=12000
  CRC = (address + cmd + speed.byte3 + speed.byte2 + speed.byte1 + speed.byte0 + |
        speed2.byte3 + speed2.byte2 + speed2.byte1 + speed2.byte0)&0x7F
  [address,cmd,speed.byte3,speed.byte2,speed.byte1,speed.byte0, | hserout
  speed2.byte3,speed2.byte2,speed2.byte1,speed2.byte0,crc] pause 4000

Mixed_Right
  CMD=37
  Speed=12000
  Speed2=-12000
  CRC = (address + cmd + speed.byte3 + speed.byte2 + speed.byte1 + speed.byte0 + |
        speed2.byte3 + speed2.byte2 + speed2.byte1 + speed2.byte0)&0x7F
  [address,cmd,speed.byte3,speed.byte2,speed.byte1,speed.byte0, | hserout
  speed2.byte3,speed2.byte2,speed2.byte1,speed2.byte0,crc] pause 4000

Mixed_Stop
  CMD=37
  Speed=0
  Speed2=0
  CRC = (address + cmd + speed.byte3 + speed.byte2 + speed.byte1 + speed.byte0 + |
        speed2.byte3 + speed2.byte2 + speed2.byte1 + speed2.byte0)&0x7F
  [address,cmd,speed.byte3,speed.byte2,speed.byte1,speed.byte0, | hserout
  speed2.byte3,speed2.byte2,speed2.byte1,speed2.byte0,crc]

stop
```

**Electrical Characteristics**

Characteristic	Rating	Min	Typ	Max
Pulse Per Second	PPS	0		8,000,000
Main Battery (B+ / B-)	VDC	6	24	30
Logic Battery (LB+ / LB-)	VDC	6	12	30
External Current Draw (BEC)	A			3A
Logic Circuit	mA		90	
Motor Current Per Channel	A		15	30
I/O Voltages	VDC	0		5
I/O Logic	TTL			5
Tempature Range	C	-40		+125

**Warranty**

Basic Micro warrants its products against defects in material and workmanship for a period of 90 days. If a defect is discovered, Basic Micro will, at our discretion, repair, replace, or refund the purchase price of the product in question. Contact us at [support@basicmicro.com](mailto:support@basicmicro.com). No returns will be accepted without the proper authorization.

**Copyrights and Trademarks**

Copyright© 2010 by Basic Micro, Inc. All rights reserved. PICmicro® is a trademark of Microchip Technology, Inc. The Basic Atom and Basic Micro are registered trademarks of Basic Micro Inc. Other trademarks mentioned are registered trademarks of their respective holders.

**Disclaimer**

Basic Micro cannot be held responsible for any incidental, or consequential damages resulting from use of products manufactured or sold by Basic Micro or its distributors. No products from Basic Micro should be used in any medical devices and/or medical situations. No product should be used in a life support situation.

**Contacts**

Email: [sales@basicmicro.com](mailto:sales@basicmicro.com)  
Tech support: [support@basicmicro.com](mailto:support@basicmicro.com)  
Web: <http://www.basicmicro.com>

**Discussion List**

A web based discussion board is maintained at <http://www.basicmicro.com>.

**Technical Support**

Technical support is made available by sending an email to [support@basicmicro.com](mailto:support@basicmicro.com). All email will be answered within 48 hours. All general syntax and programming questions, unless deemed to be a software issue, will be referred to the on-line discussion forums.

## Appendix C: Motor Specifications [25]

### Rs555 DC-motor Performance

Model	M5-RS555-12	
Operating v	: 5v - 15v	
Nominal v	: 12v	
No Load RPM	: 7750	
No Load A	: 0.4A	
Stall Torque	: 29.16 oz-in	205.9 mN-m
Stall Current	: 15A	
Kt	: 1.94 oz-in/A	13.7 mN-m/A
Kv	: 646 rpm/V	
Efficiency	: 68.5%	
RPM - Peak Eff	: 6660	
Torque - Peak Eff	: 4.76 oz-in	33.6 mN-m
Current - Peak Eff	: 2.5A	



# Appendix D: Encoder Specifications [26]



增量型旋转编码器

增量型 外径φ25 型号: A6A2  
 INCREMENTAL ROTARY ENCODERS, OUTSIDE DIAM φ25 MODEL: A6A2  
 替代型号SUBSTITUTE: E6A2

## 型号 小型编码器MINI TYPE ENCODERS

**A6A2 - CWZ 6C**

序列号  
 Sequence Number  
 外径φ25  
 outside diam φ25

- B: PNP开路输出 PNP open collector Output
- C: NPN开路输出 Open collector NPN output
- E: 电压 (NPN) 输出 Voltage output
- Gh: 互补输出 NPN Push Pull output
- Eh: 电压 (PNP) 输出 PNP Voltage output
- 1: DC5V
- 3: DC5~12V
- 5: DC12~24V
- 6: DC5~24V
- Z: 带复位相输出 + Zero Signal
- S: 单相输出 singleness Output "A"
- W: 多相输出 AB90° Phase Difference
- 绝对式编码器 •
- 增量式编码器 •



### 种类 Ordering Information

#### ◆ 本体 ABSOLUTE ROTARY ENCODERS

输出相 Output Mode	电源电压 Supply Voltage	输出状态 Output Configuration	替代型号 Substitute	分辨率 (脉冲/旋转) Resolution (p/r)					
				10	60	100	200	300	360
A相	DC 5~12V	电压输出 Voltage output	型号A6A2-CS3E	10	60	100	200	300	360
		NPN开路集电极输出 NPN output	型号A6A2-CS3C	10	60	100	200	300	360
	DC 12~24V	电压输出 Voltage output	型号A6A2-CS5E	10	60	100	200	300	360
		NPN开路集电极输出 NPN output	型号A6A2-CS5C	10	60	100	200	300	360
A相 .B相	DC 5~12V	电压输出 Voltage output	◎型号A6A2-CW3E	100	200	360	500		
		NPN开路集电极输出 NPN output	◎型号A6A2-CW3C	100	200	360	500		
	DC 12~24V	电压输出 Voltage output	◎型号A6A2-CW5E	100	200	360	500		
		NPN开路集电极输出 NPN output	◎型号A6A2-CW5C	100	200	360	500		
A相 .B相 Z相	DC 5~12V	电压输出 Voltage output	◎型号A6A2-CWZ3E	100	200	360	500		
		NPN开路集电极输出 NPN output	◎型号A6A2-CWZ3C	100	200	360	500		
	DC 12~24V	电压输出 Voltage output	◎型号A6A2-CWZ5E	100	200	360	500		
		NPN开路集电极输出 NPN output	◎型号A6A2-CWZ5C	100	200	360	500		

注: 订购时, 除型号之外, 还一定要指定“分辨率”。  
 分辨率为标准在库, 无记号的为接收订货生产。

- 适应定位的需要。  
带原点输出(Z相)型出现在生产线上。
- 也实现了更高精度的测长等。
- 追加了外径φ25, 分辨率500P/R的系列。

## 增量型旋转编码器

增量型 外径φ25 型号: A6A2  
INCREMENTAL ROTARY ENCODERS, OUTSIDE DIAM φ 25 MODEL: A6A2  
替代型号SUBSTITUTE: E6A2

外径: φ25 型号: A6A2 替代型号: E6A2

Miniature Rotary Encoder for  
Positioning in Space-Confining Areas

- Wide variety of supply voltages and output forms to match input devices
- Models with zero index function ideal for positioning applications
- High resolution models (300 or 360 pulses per revolution) substantially improve measuring accuracy
- High response frequency and noise immunity make encoders ideal for factory automation applications



### 额定/性能SPECIFICATIONS

型号MODEL	型号A6A2 -CS3E	型号A6A2 -CS3C	型号A6A2 -CS5C	型号A6A2 -CW3E	型号A6A2 -CW3C	型号A6A2 -CW5C	型号A6A2 -CWZ3E	型号A6A2 -CWZ3C	型号A6A2 -CWZ5C	
电源电压 Powersupplyvoltage	DC5V-5%~12V+10% 脉动(p-p)5%以下		DC12V-10% ~24V+15% 脉动(p-p)5% 以下	DC5V-5%~12V+10% 脉动(p-p)5%以下		DC12V-10% ~24V+15% 脉动(p-p)5% 以下	DC5V-5%~12V+10% 脉动(p-p)5%以下		DC12V-10% ~24V+15% 脉动(p-p)5% 以下	
消耗电流 Currentconsumption	30mA以下max		20mA以下max	30mA以下max		20mA以下max	50mA以下max		30mA以下max	
分辨率(脉冲/旋转) Resolution (See Note 1)	10, 20, 60, 100, 200, 300, 360, 500					10, 200, 360, 500				
输出相 Output phases	A相			A相、B相			A相、B相、Z相			
输出状态 Output form	电压输出 Voltage output		开路集电极输出 Open collector output	电压输出 Voltage output		开路集电极输出 Open collector output	电压输出 Voltage output		开路集电极输出 Open collector output	
输出容量 Output capacity	输出电阻: 2kΩ Applied voltage: 30VDC max 输出电流: 20mA以下 Sink current: 20 mA max 残留电压: 0.4V以下 Residual voltage: 0.4 V max (输出电流)20mA时		外加电压: DC30V以下 Applied voltage: 30 VDC max 同步电流: 30mA以下 Sink current: 30 mA max 残留电压: 0.4V以下 Residual voltage: 0.4 V max (同步电流30mA以下) Residual voltage: 0.4 V max	输出电阻: 2kΩ Output resistance: 2 k Ω 输出电流: 20mA以下 Sink current: 20 mA max 残留电压: 0.4V以下 Residual voltage: 0.4 V max (输出电流)20mA时		外加电压: DC30V以下 Applied voltage: 30 VDC max 同步电流: 30mA以下 Sink current: 30 mA max 残留电压: 0.4V以下 Residual voltage: 0.4 V max (同步电流30mA以下) Residual voltage: 0.4 V max	输出电阻: 2kΩ Output resistance: 2 k Ω 输出电流: 20mA以下 Sink current: 20 mA max 残留电压: 0.4V以下 Residual voltage: 0.4 V max (输出电流)20mA时		外加电压: DC30V以下 Applied voltage: 30 VDC max 同步电流: 30mA以下 Sink current: 30 mA max 残留电压: 0.4V以下 Residual voltage: 0.4 V max (同步电流30mA以下) Residual voltage: 0.4 V max	
最高应答频率 Maximum response frequency	30kHz									
输出相位差 Phase difference of output	A相、B相差 90° ±45°									
输出占空比 output wave from percentage	50±25%									
输出上升、下降时间 Output rise and fall times	1.0 μs以下 max 导线cable 500mm, 同步电流10mA (at sink current of 10 mA with 2 m cable)	1.0 μs以下, 导线长500mm, 控制输出电压5V, 负载电阻1kΩ 1.0 μs max. (at control output voltage of 5 V and load resistance of 1 k Ω with 2 m cable)	1.0 μs以下 max 导线cable 500mm, 同步电流10mA (at sink current of 10 mA with 2 m cable)	1.0 μs以下, 导线长500mm, 控制输出电压5V, 负载电阻1kΩ 1.0 μs max. (at control output voltage of 5 V and load resistance of 1 k Ω with 2 m cable)	1.0 μs以下 max 导线cable 500mm, 同步电流10mA (at sink current of 10 mA with 2 m cable)	1.0 μs以下, 导线长500mm, 控制输出电压5V, 负载电阻1kΩ 1.0 μs max. (at control output voltage of 5 V and load resistance of 1 k Ω with 2 m cable)	1.0 μs以下 max 导线cable 500mm, 同步电流10mA (at sink current of 10 mA with 2 m cable)	1.0 μs以下, 导线长500mm, 控制输出电压5V, 负载电阻1kΩ 1.0 μs max. (at control output voltage of 5 V and load resistance of 1 k Ω with 2 m cable)	1.0 μs以下 max 导线cable 500mm, 同步电流10mA (at sink current of 10 mA with 2 m cable)	1.0 μs以下, 导线长500mm, 控制输出电压5V, 负载电阻1kΩ 1.0 μs max. (at control output voltage of 5 V and load resistance of 1 k Ω with 2 m cable)

\*1. 电的最高应答转速由分辨率以及最高应答频率决定。  
电的最高应答频率转速(r/min) =  $\frac{\text{最高应答频率}}{\text{分辨率}} \times 60$   
因此, 如果旋转超过最高响应转速, 则电气上无法追踪信号。  
\*2. 对水、油没有保护作用。  
\*3. 接通电源时, 会流过约9A的冲流。(时间约0.3ms)

Note:  
◆The maximum electrical response revolution is determined by the resolution and maximum response frequency as follows:  
◆Maximum electrical response frequency (rpm)  
= Maximum response frequency ÷ resolution x 60  
◆This means that the A6B2 encoder will not operate electrically if its shaft speed exceeds the maximum electrical response revolution.

## 增量型旋转编码器

增量型 外径 $\phi 25$  型号: A6A2  
 INCREMENTAL ROTARY ENCODERS, OUTSIDE DIAM  $\phi 25$  MODEL: A6A2  
 替代型号SUBSTITUTE: E6A2

外径:  $\phi 25$  型号: **A6A2** 替代型号: **E6A2**

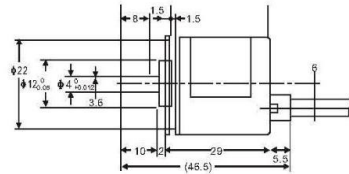
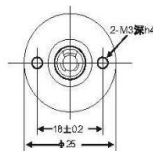
### ■ 额定/性能SPECIFICATIONS

机械性能 参数 Mechanical Spec		
起动转矩 Startion Torque		1mN·m以下Fmax
惯性力矩Moment of Inertia		$1 \times 10^{-7}$ kg·m <sup>2</sup> 以下Fmax
允许容力 Shaft loading	倾向Radial	10N
	推力Thrust	5N
允许安装精度 Mounting Tolerance	侧面误差 Ra radial: 0.03mm TIR Max; 正向误差 Axial: 0.2mm Max; 角度误差 Shaft Runout: 0.1° Max	
轴最大负荷 Allowable Shaft Load	径向Radial: 5N, 轴向 Axial: 3N	
允许最高转速Maxirun Rotating Speed	5000rpm	
环境温度 Operating Temp Range	工作时: -10 ~ +55°C 保存时: -25 ~ +80°C (不结冰)	
环境湿度Humidity	工作时、保存时: 各35 ~ 85%RH(不结露)	
绝缘电阻 Insulation resistance	20M $\Omega$ 以上(DC500V摇表)充电部整体与外壳间	
耐电压 Dielectric strength	AC500V 50/50Hz 1min 充电部整体与外壳间 充电部整体与外壳间	
振动(耐久)Vibration resistance	10~55Hz 上下振幅1.5mm X、Y、Z各方向	
冲击(耐久)Shock resistance	10~55Hz 上下振幅1.5mm X、Y、Z各方向 3次	
保护结构 Degree of protection	IEC规格 Ip50	
连接方式 Connection	导线引出型(标准导线长 cable length: 500mm)	
质量 Weight	约60g	
附件 Accessories	耦合器、伺服装置用安装配件(附于型号E6A2-CW2□)、六角扳手, 使用说明书	

### ■ 外形尺寸Dimensions (单位Unit: mm)

#### ◆ 本体

型号model:A6A2



\*PVC绝缘圆形导线 $\phi 4.5$ 芯(导体截面积: 0.15mm<sup>2</sup>、绝缘体直径:  $\phi 0.9$ mm)标准500mm长  
 Output cable (shielded) O.D.: 4 dia.  
 Standard length: 50 cm (1.64 ft)

YUEQING YING'S IMPORT&EXPORT CO., LTD

<http://www.yingselectric.com>



## 增量型旋转编码器

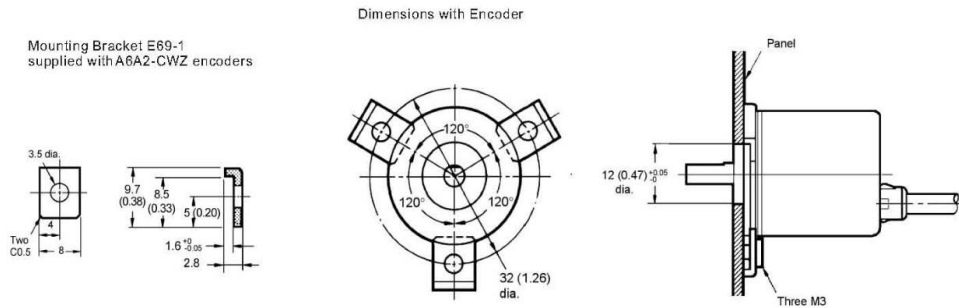
增量型 外径 $\phi 25$  型号: A6A2  
 INCREMENTAL ROTARY ENCODERS, OUTSIDE DIAM  $\phi 25$  MODEL: A6A2  
 替代型号SUBSTITUTE: E6A2

外径:  $\phi 25$  型号: A6A2 替代型号: E6A2

### ■ 输出段回路图OUTPUT CIRCUIT DIAGRAMS

型号 MODEL	输出回路 Output diagram	输出方式 Output wave	连接 Wire Code												
型号A6A2-CS3C 型号A6A2-CS5C		<b>输出晶体管Output transistor</b> 	<table border="1"> <thead> <tr> <th>线色Wire color</th> <th>内容Signal</th> </tr> </thead> <tbody> <tr> <td>棕Brown</td> <td>+VCC</td> </tr> <tr> <td>黑Black</td> <td>A相ph</td> </tr> <tr> <td>白White</td> <td>B相ph</td> </tr> <tr> <td>橙Orange</td> <td>Z相ph</td> </tr> <tr> <td>蓝Blue</td> <td>0V</td> </tr> </tbody> </table>	线色Wire color	内容Signal	棕Brown	+VCC	黑Black	A相ph	白White	B相ph	橙Orange	Z相ph	蓝Blue	0V
线色Wire color		内容Signal													
棕Brown	+VCC														
黑Black	A相ph														
白White	B相ph														
橙Orange	Z相ph														
蓝Blue	0V														
型号A6A2-CW3C 型号A6A2-CW5C	<b>型号A6A2-CW3C</b> <b>型号A6A2-CW5C</b> 	<b>旋转方向: CW</b> Direction of rotation: CW (从轴方向观察为右转) Clockwise as viewed from the shaft <b>输出晶体管</b> 	<p>注: 1. 单型(型号A6A2-CS□□)中, 白和橙色无输出。(未连接)                      2. 双向型(型号A6A2-CW□□)中, 棕色无输出。(未连接)                      3. 电压输出型中, 可吸收20mA的电流。</p> <p>注1* (B) (L) 表示电压输出型的状态。                      2. 顺时针(CW)旋转时, A相比B相, 超前1/4T ± 1/8T相位。逆时针(CCW)旋转时, A相比B相滞后1/4T ± 1/8T相位。</p> <p>Note:                      1. The white (green) and orange (yellow) lines of the single type (E6A2-CS) do not output signals (no connection).                      2. The orange (yellow) line of the reversible type (E6A2-CW) does not output signal (no connection).                      3. The voltage output type is capable of sinking a maximum of 20 mA.</p>												
型号A6A2-CW3E 型号A6A2-CWZ3E		<b>旋转方向: CCW</b> Direction of rotation: CCW (从轴方向观察为右转) Counterclockwise as viewed from the shaft <b>输出晶体管</b> 	<p>注: 1. * (E) and (L) indicate the output levels of the voltage output type.                      2. Output A leads B by 1/4T ± 1/8T when the shaft revolves clockwise. Output A lags behind B by 1/4T ± 1/8T when the shaft revolves counterclockwise.</p> <p>Note:                      1. The white (green) and orange (yellow) lines of the single type (E6A2-CS) do not output signals (no connection).                      2. The orange (yellow) line of the reversible type (E6A2-CW) does not output signal (no connection).                      3. The voltage output type is capable of sinking a maximum of 20 mA.</p>												
型号A6A2-CS3E		<b>输出晶体管</b> 													

### ■ 安装尺寸 Installation Dimension (单位Unit: mm)

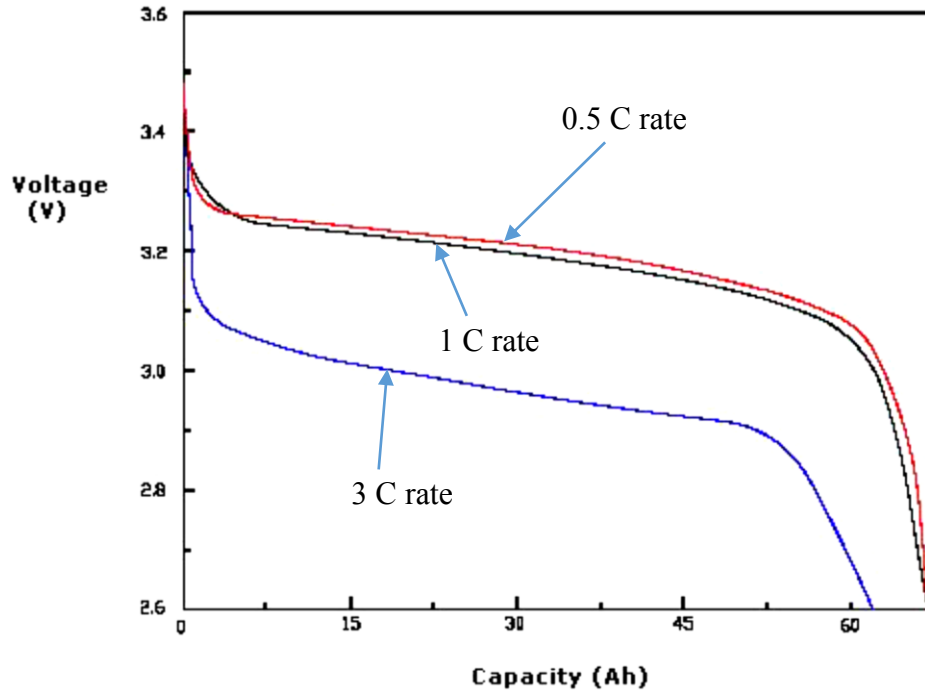


## Appendix E: Battery

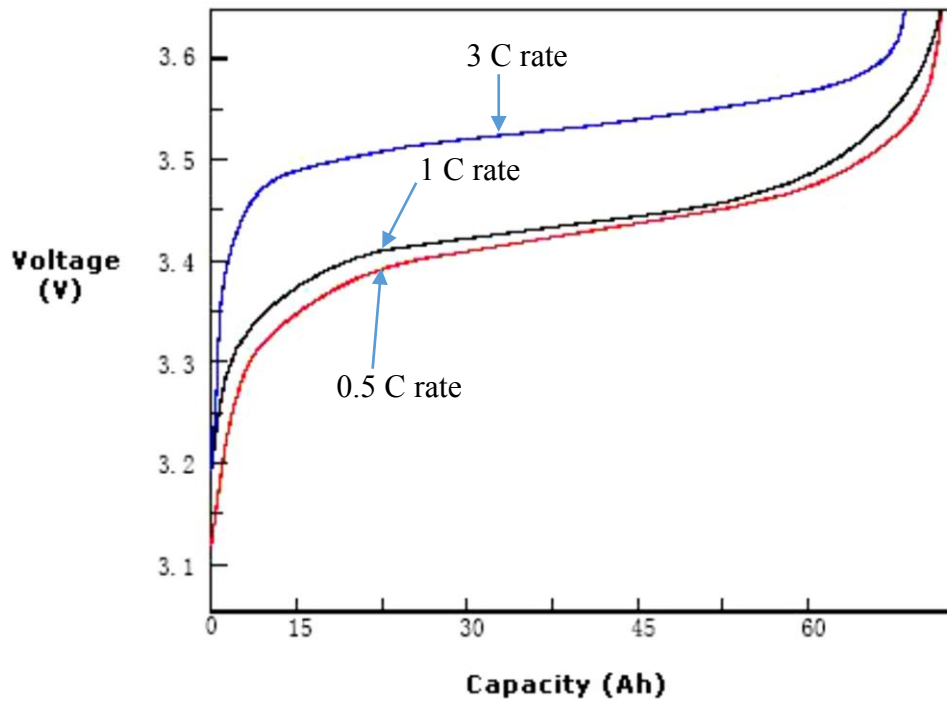
### Battery Specifications [53]

Nominal Capacity	60Ah	Working Voltage	Single cell charging: 3.8V
			Battery pack charging: 3.65V
			Single cell charging: 2.5V
			Battery pack discharging: 2.8V
Max. Charging Current	$\leq 3C$	Max. Discharging Current	Continuous Current: $\leq 3C$
			Impulse Current: $\leq 10C$
Standard Charging Current	0.3~0.8C	Best Charging Current	0.5C
Internal Resistance	$\leq 2.0m\Omega$	Cycle Life	Single cell $\geq 2000$ times (80%DOD)
			Battery pack $\geq 1500$ times (80%DOD)
Temp. resistance of Shell	$\leq 135^\circ C$	Working Temperature	Charging: $> 0^\circ C$
			Discharging: $-20^\circ C - 65^\circ C$
Self Discharge rate ( month)	$\leq 3\%$	Cell Weight	$2.5kg \pm 100g$
Energy Density	85~100Wh/kg	Power Density	$> 800w/kg$
Dimension	126mm*65mm*180mm		

Table E. 1: Specification of LiFePO4 Prismatic Module: 3.2V 60 Ah, 10C Rate (192 wh).



E. 1: Discharge curve. 0.5 C rate means that the discharge current equals half the battery capacity. In our case, the battery capacity is 60 Ah so the top curve is when the discharge current is 30 A.



E. 2: Charging curve. 0.5 C rate means that the charging current equals half the battery capacity. In our case, the battery capacity is 60 Ah so the top curve is when the discharge current is 30 A.

## Material Safety Data Sheet [54]

### Section 1 - Chemical Product and Company Identification

**Product Name:** Lithium Iron Manganese Phosphate Battery (GBS-LFMP60AH)

**Sample Code:** GBS-LFMP60AH

**Manufacture:** ZHEJIANG GBS ENERGY CO., LTD

**Address:** No.6 BEIHUAN EAST ROAD, YUYAO, ZHEJIANG PROVINCE.

**Post Code:** 315410

**Tel:** 0574-62655559

**Emergency Tel (Within USA and Canada):** CHEMTREC 1-800-424-9300

**Emergency Tel (Outside USA and Canada) for Shipment to USA:** CHEMTREC +1 703-527-3887

**Fax:** 0574-62655552

**Email:** [Sharon@gbsystem.com.cn](mailto:Sharon@gbsystem.com.cn)

### Section 2 – Composition/Information on Ingredient

Chemical Name	Chemical Formula or Abbreviation	CAS No.	In % By Weight
Lithium Iron Manganese Phosphate	LiFeMnPO <sub>4</sub>	---	38.1
Graphite	C	7782-42-5	18.1
Aluminum	Al	7429-90-5	7.6
Copper	Cu	7440-50-8	11.4
Diaphragm paper (PP)	(C <sub>3</sub> H <sub>6</sub> ) <sub>n</sub>	9003-07-0	4.5
Electrolyte (Lithium hexafluorophosphate)	LiPF <sub>6</sub>	21324-40-3	20.3

## Section 3 - Hazards Identification

### **Routes of Entry**

Inhalation, Skin, Ingestion.

### **Health Hazards (Acute and Chronic)**

These chemicals are contained in a sealed can. Risk of exposure occurs only if the battery is mechanically or electrically abused. The most likely risk is acute exposure when a battery vents.

### **Sign/Symptoms of Exposure**

May be a reproductive hazard. Lithium can cause thermal and chemical burns upon contact with the skin.

### **Medical Conditions Generally Aggravated by Exposure**

An acute exposure will not generally aggravate any medical condition.

## Section 4 - First Aid Measures

### **Eye**

Flush eyes with plenty of water for at least 15 minutes, occasionally lifting the upper and lower eyelids. Get medical aid.

### **Skin**

Remove contaminated clothes and rinse skin with plenty of water or shower for 15 minutes. Get medical aid.

### **Inhalation**

Remove from exposure and move to fresh air immediately. Use oxygen if available.

### **Ingestion**

Give at least 2 glasses of milk or water. Induce vomiting unless patient is unconscious. Call a physician.

## Section 5 - Fire Fighting Measures

**Flash Point:** N/A.

**Auto-Ignition Temperature:** N/A. **Extinguishing Media** Dry chemical, CO<sub>2</sub>.

### **Firefighting**

In case of fire in an adjacent area, use CO<sub>2</sub> or dry chemical extinguisher if batteries in their original containers since the fuel of the fire is basically paper products. For bulk quantities of unpackaged batteries use appropriate method. In this case, do not use water.



## Section 6 - Accidental Release Measures

### **Steps to be Taken in case Material is Released or Spilled**

If the battery is accidentally broken and organic electrolyte leaks out, wipe it up with a cloth, and dispose of it in a plastic bag and put into a steel can. The preferred response is to leave the area and allow the batteries to cool and vapors to dissipate. Provide maximum ventilation. Avoid skin and eye contact or inhalation of vapors. Remove spilled liquid with absorbent and incinerate.

### **Waste Disposal Method**

It is recommended to discharge the battery to the end, to use up the metal lithium inside the battery, and to bury the discharged battery in soil.

## Section 7 - Handling and Storage

The batteries should not be opened, destroyed or incinerate, since they may leak or rupture and release to the environment the ingredients that they contain in the hermetically sealed container. Do not short circuit terminals, or over charge the battery, forced over-discharge, throw to fire. Do not crush or puncture the battery, or immerse in liquids.

### **Precautions to be taken in handling and storing**

Avoid mechanical or electrical abuse. Storage preferably in cool, dry and ventilated area, which is subject to little temperature change. Storage at high temperatures should be avoided.

Do not place the battery near heating equipment, nor expose to direct sunlight for long periods.

### **Other Precautions**

Batteries may explode or cause burns, if disassembled, crushed or exposed to fire or high temperatures. Do not short or install with incorrect polarity.

## Section 8 - Exposure Controls, Personal Protection

### **Respiratory Protection**

In case of battery venting, provide as much ventilation as possible. Avoid confined areas with venting batteries. Respiratory Protection is not necessary under conditions of normal use.

### **Ventilation**

Not necessary under conditions of normal use.

### **Protective Gloves**

Not necessary under conditions of normal use.

**Other Protective Clothing or Equipment** Not necessary under conditions of normal use.

### **Personal Protection is recommended for venting batteries**

Respiratory Protection, Protective Gloves, Protective Clothing and safety glass with side shields.

## Section 9 - Physical and Chemical Properties

**Nominal Voltage:** 3.2V **Rated Capacity:** 60Ah.

**Appearance Characters:** Cyan, odorless, quadrate battery.

**Chemical Uses:** ELECTRIC MOTORCYCLE AND ELECTRIC VEHICLE.

## Section 10 - Stability and Reactivity

### Stability

Stable

### Conditions to Avoid

Heating, mechanical abuse and electrical abuse.

**Hazardous Decomposition Products** N/A.

**Hazardous Polymerization** N/A.

If leaked, forbidden to contact with strong oxidizers, mineral acids, strong alkalies, halogenated hydrocarbons.

## Section 11 - Toxicological Information

Inhalation, skin contact and eye contact are possible when the battery is opened. Exposure to internal contents, the corrosive fumes will be very irritating to skin, eyes and mucous membranes. Overexposure can cause symptoms of non-fibrotic lung injury and membrane irritation.

## Section 12 - Ecological Information

When promptly used or disposed the battery does not present environmental hazard. When disposed, keep away from water, rain and snow.

## Section 13 - Disposal Considerations

### APPROPRIATE METHOD OF DISPOSAL OF SUBSTANCE OR PREPARATION

If batteries are still fully charged or only partially discharged, they can be considered a reactive hazardous waste because of significant amount of unreacted, or unconsumed lithium remaining in the spent battery. The batteries must be neutralized through an approved secondary treatment facility prior to disposal as a hazardous waste. Recycling of battery can be done in authorized facility, through licensed waste carrier.

## Section 14 - Transport Information

The Lithium Iron Manganese Phosphate Battery (GBS-LFMP60AH) have pass the test UN38.3, according the report ID: 1001043-016 and ID: 1001043-016a. According to PACKING INSTRUCTION 965 ~ 970 of IATA DGR 51<sup>st</sup> Edition for transportation.

**Electric energy is 192Wh. Watt-hour exceeds the standard, so it belongs to dangerous goods. Cargo only.**

**IATA Proper Shipping Name: Lithium ion batteries+(including lithium polymer batteries)**

**Hazard Class: 9**

**Identification Number: UN3480**

**Packaging group: II**

**Maritime Transport IMDG: Lithium ion batteries+(including lithium polymer batteries)**

**IMDG Class: 9**

**UN Number: UN3480**

**Packaging group: II**

More information concerning shipping, testing, marking and packaging can be obtained from Labelmaster at <http://www.labelmaster.com>.

Separate Li-ion batteries when shipping to prevent short-circuiting. They should be packed in strong packaging for support during transport.

In the case of transportation, confirm no leakage and no overspill from a container. Take in a cargo of them without falling, dropping and breakage. Prevent collapse of cargo piles and wet by rain. The container must be handled carefully. Do not give shocks that result in a mark of hitting on a cell. Please refer to Section 7-HANDLING AND STORAGE also.

## Section 15 - Regulatory Information

### Law Information

《Dangerous Goods Regulation》

《Recommendations on the Transport of Dangerous Goods Model Regulations》

《International Maritime Dangerous Goods》

《Classification and code of dangerous goods》

OSHA Hazard Communication Standard Status

Toxic Substances Control Act (TSCA) Status

SARA Title III

RCRA

In accordance with all Federal, State and Local laws.

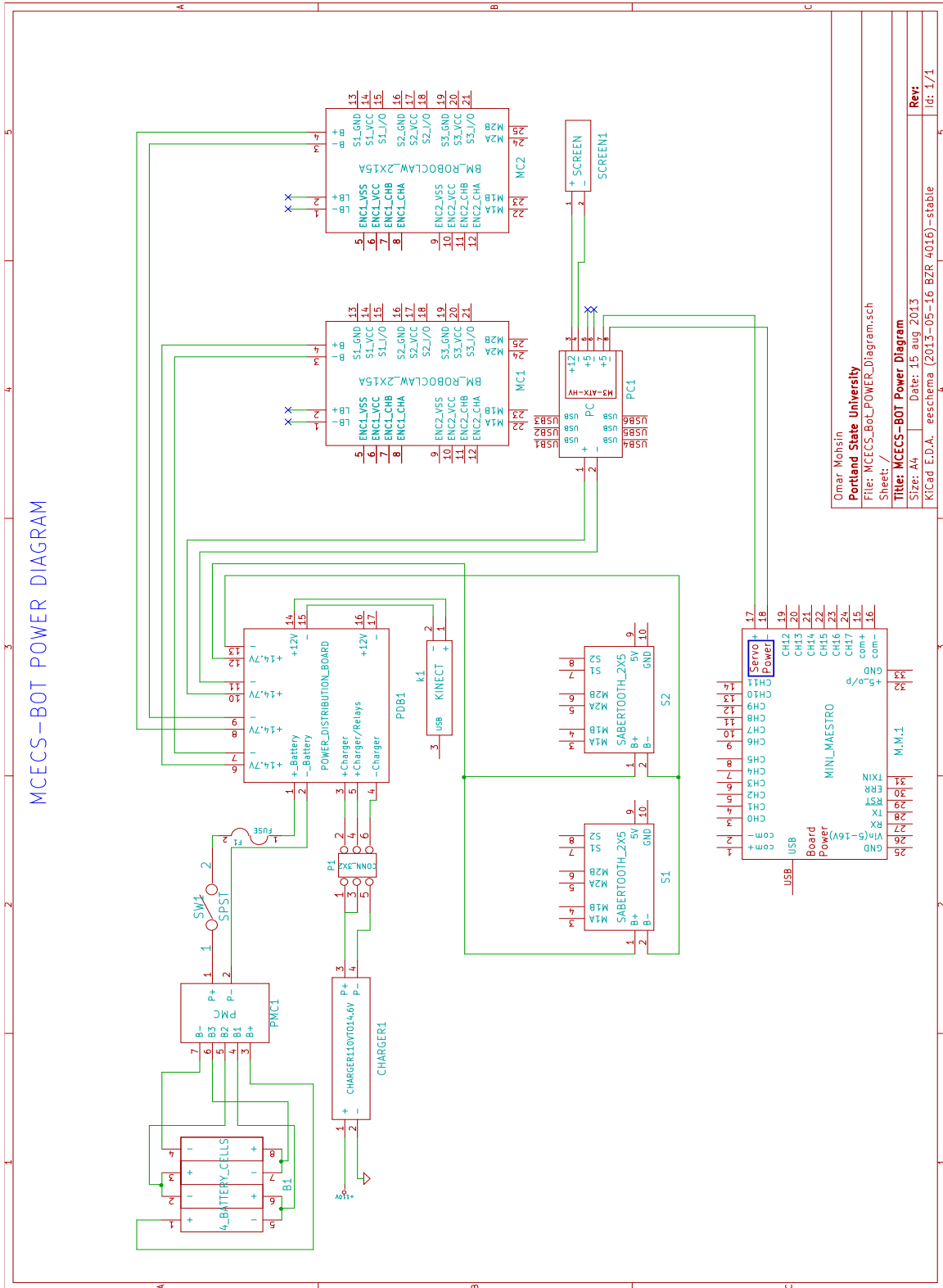
## Section 16 - Additional Information

The above information is based on the data of which we are aware and is believed to be correct as of the data hereof. Since this information may be applied under conditions beyond our control and with which may be unfamiliar and since data made available subsequent to the data hereof may suggest modifications of the information, we do not assume any responsibility for the results of its use. This information is furnished upon condition that the person receiving it shall make his own determination of the suitability of the material for his particular purpose.

MSDS Creation Date: January 22, 2010

MSDS Revision Date: May 11, 2010

# Appendix F: System Circuit Boards Diagram



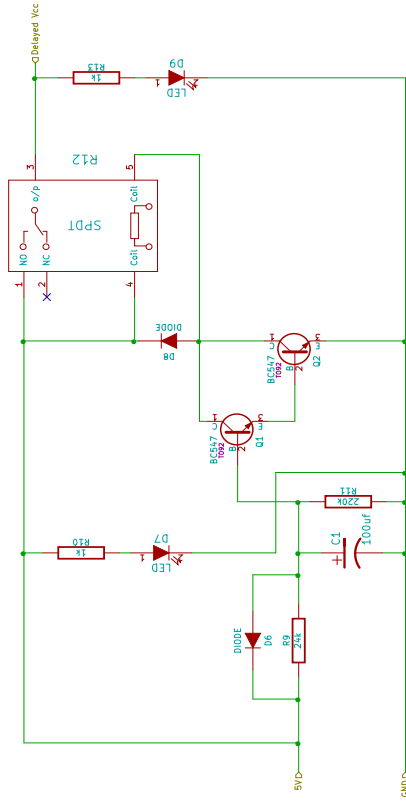
MCECS-BOT POWER DIAGRAM

Omar Mohsin
Portland State University
File: MCECS_Bot_POWER_Diagram.sch
Sheet: /
Title: MCECS-BOT Power Diagram
Size: A4
Date: 15 aug 2013
KiCad E.D.A. eeschema (2013-05-16 BZR #016)-stable
Rev: /
Id: 1/1





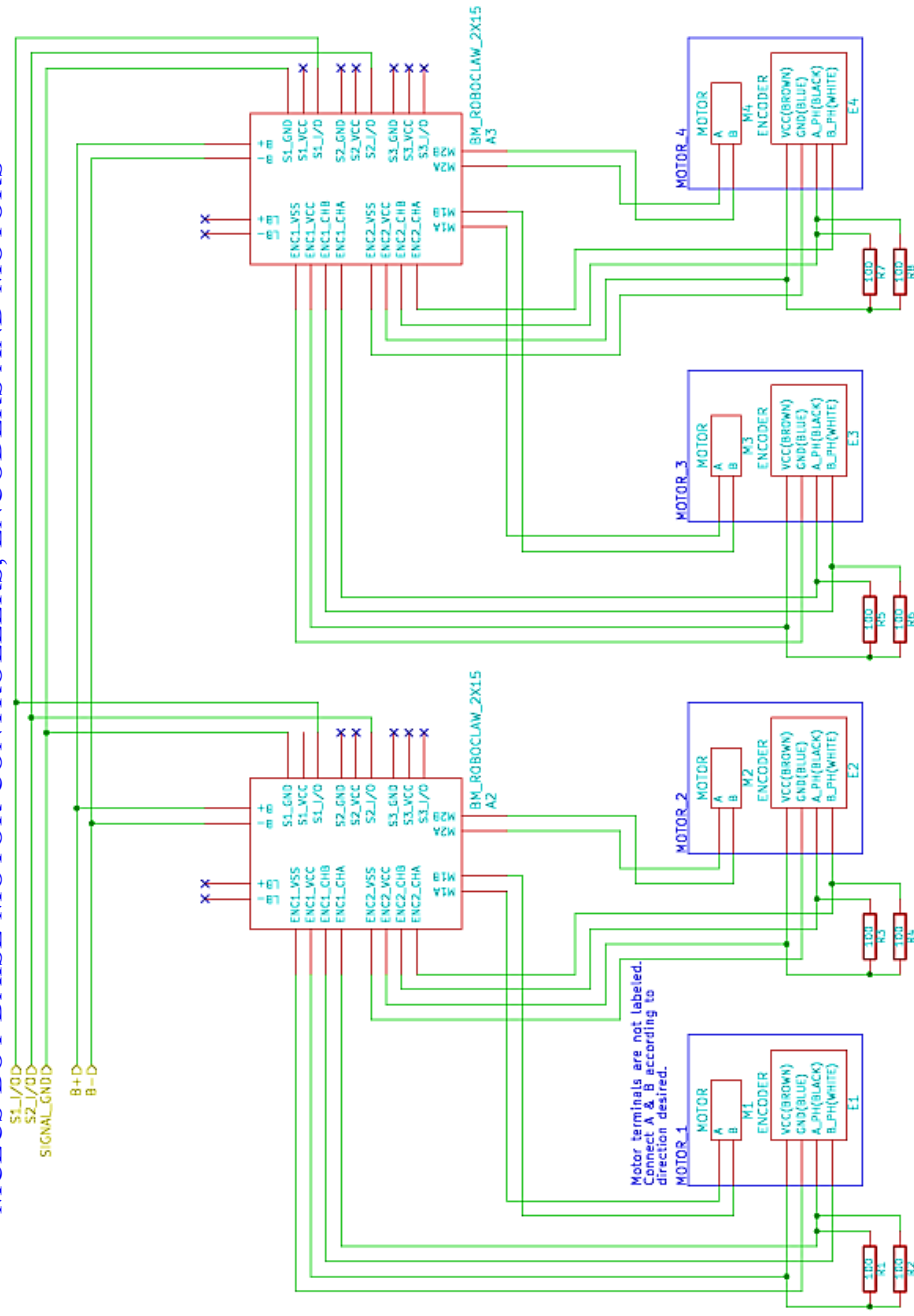
# Delay Circuit



Omar Mohsin	
Portland State University	
File: Delay Circuit.sch	
Sheet: /Delay Circuit/	
Title: Delay Circuit	
Size: A4	Date: 24 aug 2013
KiCad E.D.A.	eschema (2013-05-16 BZR 4016) - stable
	Rev: 1
	Id: 3/3



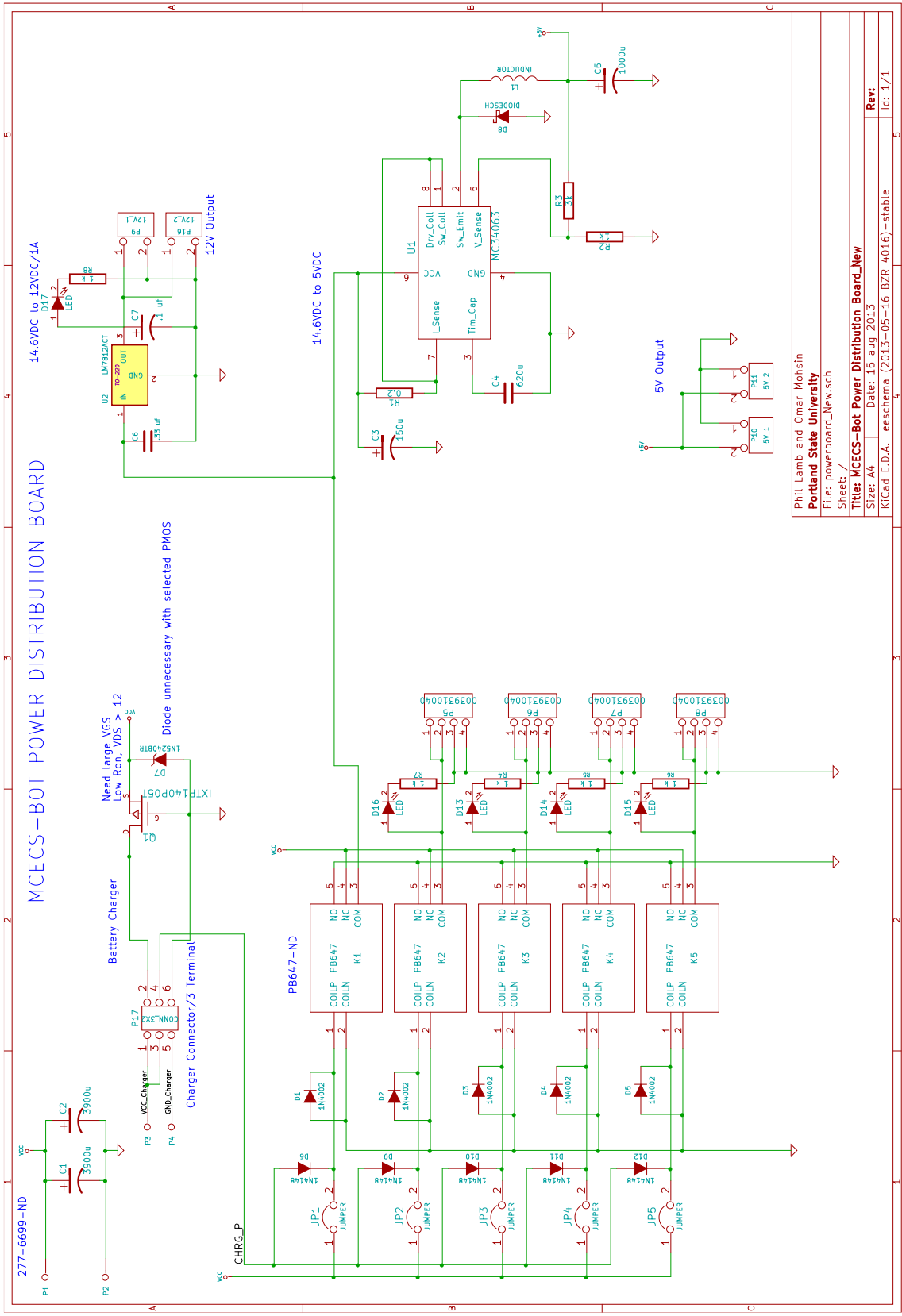
# MCECS-BOT BAISE MOTOR CONTROLLERS, ENCODERS AND MOTORS



Motor terminals are not labeled.  
Connect A & B according to  
direction desired.

Encoder phases must match direction of motor.  
if backwards internal counter in RobboClaw will  
count down when moving forward.

File: Motor_Control.sch
Sheet: /Motor_Control/
Title:
Size: A4
Date: 9 dec 2012
KiCad E.D.A.
Rev: 1
1 of 3

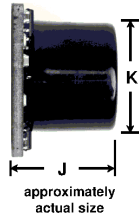
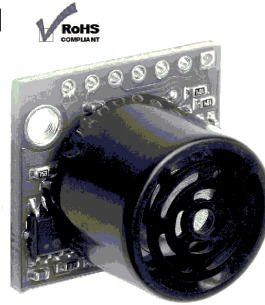


MCECS-BOT POWER DISTRIBUTION BOARD

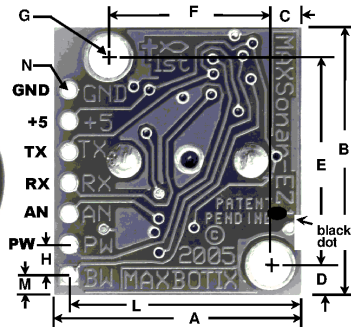
Phil Lamb and Omar Mohsin	
Portland State University	
File: powerboard_New.sch	
Sheet: /	
Size: A4	Date: 15 aug 2013
KiCad E.D.A. eeschema (2013-06-16 BZR-4016)-stable	
<b>Rev:</b>	Id: 1/1

# LV-MaxSonar®-EZ0™ High Performance Sonar Range Finder

With 2.5V - 5.5V power the LV-MaxSonar®-EZ0™ provides very short to long-range detection and ranging, in an incredibly small package. The LV-MaxSonar®-EZ0™ detects objects from 0-inches to 254-inches (6.45-meters) and provides sonar range information from 6-inches out to 254-inches with 1-inch resolution. Objects from 0-inches to 6-inches typically range as 6-inches. The interface output formats included are pulse width output, analog voltage output, and serial digital output.



approximately actual size



A	0.785"	19.9 mm	H	0.100"	2.54 mm
B	0.870"	22.1 mm	J	0.610"	15.5 mm
C	0.100"	2.54 mm	K	0.645"	16.4 mm
D	0.100"	2.54 mm	L	0.735"	18.7 mm
E	0.670"	17.0 mm	M	0.065"	1.7 mm
F	0.510"	12.6 mm	N	0.038" dia.	1.0 mm dia.
G	0.124" dia.	3.1 mm dia.	weight, 4.3 grams		

values are nominal

## Features

- Continuously variable gain for beam control and side lobe suppression
- Object detection includes zero range objects
- 2.5V to 5.5V supply with 2mA typical current draw
- Readings can occur up to every 50mS, (20-Hz rate)
- Free run operation can continually measure and output range information
- Triggered operation provides the range reading as desired
- All interfaces are active simultaneously
- Serial, 0 to Vcc, 9600Baud, 81N
- Analog, (Vcc/512) / inch
- Pulse width, (147uS/inch)
- Learns ringdown pattern when commanded to start ranging
- Designed for protected indoor environments
- Sensor operates at 42KHz
- High output square wave sensor drive (double Vcc)

## Benefits

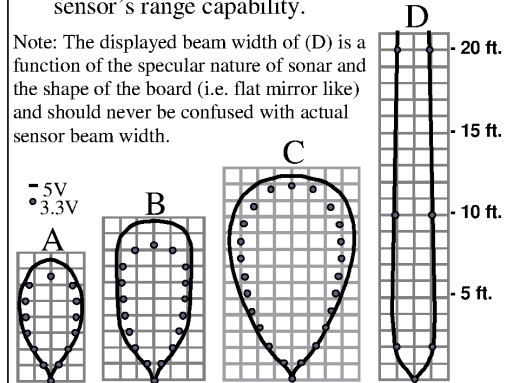
- Very low cost sonar ranger
- Reliable and stable range data
- Sensor dead zone virtually gone
- Lowest power ranger
- Quality beam characteristics
- Mounting holes provided on the circuit board
- Very low power ranger, excellent for multiple sensor or battery based systems
- Can be triggered externally or internally
- Sensor reports the range reading directly, frees up user processor
- Fast measurement cycle
- User can choose any of the three sensor outputs

## Beam Characteristics

The LV-MaxSonar®-EZ0™ has the most sensitivity of the LV-MaxSonar®-EZ0™ product line, yielding a controlled wide beam with high sensitivity. Sample results for measured beam patterns are shown below on a 12-inch grid. The detection pattern is shown for;

- (A) 0.25-inch diameter dowel, note the narrow beam for close small objects,
- (B) 1-inch diameter dowel, note the long narrow detection pattern,
- (C) 3.25-inch diameter rod, note the long controlled detection pattern,
- (D) 11-inch wide board moved left to right with the board parallel to the front sensor face and the sensor stationary. This shows the sensor's range capability.

Note: The displayed beam width of (D) is a function of the specular nature of sonar and the shape of the board (i.e. flat mirror like) and should never be confused with actual sensor beam width.



beam characteristics are approximate



## Appendix H: The LiFePo4 Battery Charger [37]

### Features:

- Intelligent charger designed for 4 cell LiFePO4 battery packs.
- CPU control and pulse width modulation (PWM) technology, charging current and output voltage is controlled accurately to ensure fully-charged and avoid over-charging.
- Built-in cooling fan to ensure charger long service life.
- Safety protection:
  - Over Voltage Protection.
  - Short Circuit Protection.
  - Output Reverse Protection
- Charging time:
  - Charging Time= (1.41 \* Ah rate of the pack) / 10A charge current.
- Built-in IC to cut-off power automatically when battery is fully charged.
- LED indicators:
  - LED 1 = Red = Power On.
  - LED 2 = Red = Charging.
  - LED 3 = Green = Fully Charged

### User Instructions:

- Check the output plug of this charger and make sure it is not loose.
- Before charging, please connect the charger's alligator clips to battery first: the "red" clip connects to the POSITIVE pole and the "black" connects to the NEGATIVE pole. After, connect the input power plug to the indoor power supply.
- The charger applies the intelligent charging method of constant current and constant voltage. The charger will automatically shut off when battery is fully charged. Unplug the input power supply then disconnect the output clips.
- When charger is not in use or finished charging, be sure to unplug input power supply.
- Indicator LED instruction:
  - "Charging" indicator shows "red" under normal charging state. It turns into "Green" when the battery is fully charged, the battery can be put into use at this time.
  - The charging voltage of this charger is 12V, and shut-off current is 1A. It can only be operated when connect with not fully charged battery.

Includes:

- Battery Charger.
- Power Cable.
- Manual

Technical Specifications:

Item	4 Cell 14.6V (12.8V Nominal) 10A Charger
Model	TN1210JL
Max Output Power	240W
Output Voltage	14.6V $\pm$ 0.02Vdc
Output Current	10A
Rated Input Voltage	110Vac
Input Voltage	AC90-135V
Constant Voltage	14.6V $\pm$ 0.2V
AC Input Voltage Frequency	50 $\zeta$ 60 Hz
Constant Current	10A $\pm$ 1A
Constant Voltage	14.6 Vdc, 10A $\zeta$
Shut-off Current	1A
Power Efficiency	90% (Vin=110Vac, rated load)
Over Voltage Protection	YES
Software Over Voltage Protection	The charger software limits the maximum output voltage to a level suitable for the connected battery system
Thermal Protection	N/A
Current Limiting Protection	YES (At CC Mode)
Reverse Polarity Protection	When output wires are reversely connected to the battery the charger will not operate and will work normally when DC wires are correctly connected
Electric Strength Test Input-Output	1500Vac/10mA/1 min (No Breakdown)
Isolation Resistance Input-ground	10m $\zeta$ @500Vdc
Isolation Resistance Output-ground	10m $\zeta$ @500Vdc
Leakage Current	<3.5mA
Safety	CE/UL Compliant
High Temperature Ambient Operating	+40°C
Low Temperature	-10°C
High Temperature Storage	+70°C (Normal after recovery under normal temperature for 2 hours)

Low Temperature Storage	+40°C (Normal after recovery under normal temperature for 2 hours)
Random Vibration	20Hz to 2000Hz 3 Grms 20 hours per axis
Repetitive Shock	40g peak 3 orthogonal axes, 3+ and 3- in each axis, 11ms pulse width
Thermal Shock	-35°C to 75°C, <3min transition, 2.5 hours dwell, 200 cycle
Drop Test	BS EN60068-2-32:1993 TEST ED: free fall appendix B
AC Wire Length	1.5 m length
DC Wire Length	1.5 m length
Dimensions	173mmX88mmX62mm (L*W*H)
Net Weight	1.0Kg
Output Connector Type	XLR Connector
Casing	Aluminum

The charger has been calibrated to take account of the voltage drop in the DC output cables during operation, to prevent the possibility of over or under charging of the battery it is recommended the DC output cables are connected directly to the battery without modification.

Warning:

- Only for use with LiFePO4 4 cell battery pack (If used with battery pack of less cells will cause damage to battery and possible personal injury may occur).
- Unplug charger after use. x To prevent electrical shock, DO NOT remove casing
- Be careful of high-voltage inside charger. If failure happens, please contact us. Users and non-professional technicians are prohibited to open the charger!
- Keep charger out of children's reach. x The charger and battery belongs in indoor environment with good ventilation and good cooling system, Do not expose to humid, high temperature, flammable, explosive gas environments.
- Do not bring charger along during transportation to prevent shock damage! x Read the instructions carefully before using. x Damage caused by improper use is not covered by warranty.

## Appendix I: Code

This appendix contains different codes that have been used and developed by different teams that have been working on the MCECS-Bot in the last two years.

### Arduino Uno Code

The code here is responsible for communicating with Arduino Mega 1 and send the sonars measurements to it. As can be seen in Appendix F, Arduino Uno are connected to Arduino Mega 1 and works as slave to it. The code is shown below:

```
// Slave Sonar
// Jesse Adams
// Rev 2. Omar Mohsin
#include <Wire.h>
#define arraysize 12 // size of array that stores sonar data

uint8_t pulse1;
uint8_t pulse2;
uint8_t pulse3;
uint8_t pulse4;
uint8_t pulse5;
uint8_t pulse6;
uint8_t pulse7;
uint8_t pulse8;
uint8_t pulse9;
uint8_t pulse10;
uint8_t pulse11;
uint8_t pulse12;

uint8_t sonar1 = 7;
uint8_t sonar2 = 6;
uint8_t sonar3 = 5;
uint8_t sonar4 = 2;
uint8_t sonar5 = 3;
uint8_t sonar6 = 4;
uint8_t sonar7 = 10;
uint8_t sonar8 = 9;
uint8_t sonar9 = 8;
uint8_t sonar10 = 11;
uint8_t sonar11 = 12;
uint8_t sonar12 = 13;
int InterruptPin = A1;
int txline = 0;
```



```

boolean ALERT = false;    // Tells whether object detect. Must "alert" the mega
uint8_t obstacle;        // Holds value for data transmission to mega
uint8_t data_to_send[2]; // In I2C protocol, if sending multiple bytes, they
                        // must be stored in a uint8_t array
int threshold = 15;      // distance in inches that will trigger robot to stop
int kinectVal = -1;      // initializing value. ie don't do anything until instructed
uint8_t rangevalue[arraysize] = {0,0,0,0,0,0,0,0,0,0,0}; // data array for sonars
int slave_address = 2;   // I2C slave address of Arduino

void setup()
{
  Wire.begin(slave_address); // join i2c bus with address #2
  Wire.onReceive(receiveEvent); // register event
  Wire.onRequest(requestEvent); // register event
  Serial.begin(9600); // start serial for output
  pinMode(InterruptPin, OUTPUT); // Alert pin for Mega
  digitalWrite(InterruptPin, ALERT); // Initialize as low

  delay(250); // delay required for sonars

  pulseIn(sonar1, HIGH); // Each sonar must be initialized with a read of 102ms
  delay(102); // this is a calibration reading
  pulseIn(sonar2, HIGH);
  delay(102);
  pulseIn(sonar3, HIGH);
  delay(102);
  pulseIn(sonar4, HIGH);
  delay(102);
  pulseIn(sonar5, HIGH);
  delay(102);
  pulseIn(sonar6, HIGH);
  delay(102);
  pulseIn(sonar7, HIGH);
  delay(102);
  pulseIn(sonar8, HIGH);
  delay(102);
  pulseIn(sonar9, HIGH);
  delay(102);
  pulseIn(sonar10, HIGH);
  delay(102);
  pulseIn(sonar11, HIGH);
  delay(102);
  pulseIn(sonar12, HIGH);
  delay(102);
  Serial.begin(9600);
  analogWrite(txline, 255);
  delayMicroseconds(20);
  analogWrite(txline, 0);
}

void loop()
{
  // Code that executes while waiting from I2C communication

```

```

// This commented part is part of Jesse's original code
if(Serial.available()){ // If new movement available, retrieve.
    kinectVal = Serial.read();
    Serial.print("Kinectval: ");
    Serial.println(kinectVal);
}

int pulse1,pulse2,pulse3,pulse4,pulse5,pulse6,pulse7,pulse8,pulse9,pulse10,pulse11,pulse12;
int distance1,distance2,distance3,distance4,distance5,distance6,distance7,distance8,distance9, distance10,
distance11, distance12;

pulse1 = pulseIn(sonar1, HIGH);
pulse2 = pulseIn(sonar2, HIGH);
pulse3 = pulseIn(sonar3, HIGH);
pulse4 = pulseIn(sonar4, HIGH);
pulse5 = pulseIn(sonar5, HIGH);
pulse6 = pulseIn(sonar6, HIGH);
pulse7 = pulseIn(sonar7, HIGH);
pulse8 = pulseIn(sonar8, HIGH);
pulse9 = pulseIn(sonar9, HIGH);
pulse10 = pulseIn(sonar10, HIGH);
pulse11 = pulseIn(sonar11, HIGH);
pulse12 = pulseIn(sonar12, HIGH);

distance1 = pulse1/147;
distance2 = pulse2/147;
distance3 = pulse3/147;
distance4 = pulse4/147;
distance5 = pulse5/147;
distance6 = pulse6/147;

distance7 = pulse7/147;
distance8 = pulse8/147;
distance9 = pulse9/147;
distance10 = pulse10/147;
distance11 = pulse11/147;
distance12 = pulse12/147;

rangevalue[0] = distance1;
rangevalue[1] = distance2;
rangevalue[2] = distance3;
rangevalue[3] = distance4;
rangevalue[4] = distance5;
rangevalue[5] = distance6;
rangevalue[6] = distance7;
rangevalue[7] = distance8;
rangevalue[8] = distance9;
rangevalue[9] = distance10;
rangevalue[10] = distance11;
rangevalue[11] = distance12;
/*Serial.print(distance1);
Serial.print(",");
Serial.print(distance2);

```

```

Serial.print(",");
Serial.print(distance3);
Serial.print(",");
Serial.print(distance4);
Serial.print(",");
Serial.print(distance5);
Serial.print(",");
Serial.print(distance6);
Serial.print(",   ");

Serial.print(distance7);
Serial.print(",");
Serial.print(distance8);
Serial.print(",");
Serial.print(distance9);
Serial.print(",");
Serial.print(distance10);
Serial.print(",");
Serial.print(distance11);
Serial.print(",");
Serial.println(distance12);
*/

```

```

switch(kinectVal){

  case 48:           // case 0: Turn Left
    Serial.print(distance1);
    Serial.print(",");
    Serial.print(distance2);
    Serial.print(",");
    Serial.print(distance3);
    Serial.print(",");
    Serial.print(distance4);
    Serial.print(",");
    Serial.print(distance5);
    Serial.print(",");
    Serial.print(distance6);
    Serial.print(",   ");

    Serial.print(distance7);
    Serial.print(",");
    Serial.print(distance8);
    Serial.print(",");
    Serial.print(distance9);
    Serial.print(",");
    Serial.print(distance10);
    Serial.print(",");
    Serial.print(distance11);
    Serial.print(",");
    Serial.println(distance12);

```

```

break;

/*case 1:          // case 1: Turn Right
Serial.println("Case 1");
pulse3 = getSonar(sonar3);
if(pulse3 < threshold){
    stop_robot(sonar3, pulse3);
}

pulse5 = getSonar(sonar5);
if(pulse5 < threshold){
    stop_robot(sonar5, pulse5);
}

pulse8 = getSonar(sonar8);
if(pulse8 < threshold){
    stop_robot(sonar8, pulse8);
}
pulse11 = getSonar(sonar11);
if(pulse11 < threshold){
    stop_robot(sonar11, pulse11);
}
break;
*/
/* case 2:          // case 2: Move Forward
Serial.println("Case 2");
pulse1 = getSonar(sonar1);
if(pulse1 < threshold){
    stop_robot(sonar1, pulse1);
}
pulse2 = getSonar(sonar2);
if(pulse2 < threshold){
    stop_robot(sonar2, pulse2);
}
pulse11 = getSonar(sonar11);
if(pulse11 < threshold){
    stop_robot(sonar11, pulse11);
}
//pulse12 = getSonar(sonar12);
// if(pulse12 < threshold){
//     stop_robot(sonar12, pulse12);
// }

break;

case 3:          // case 3: Move Backward
Serial.println("Case 3");
pulse6 = getSonar(sonar6);
if(pulse6 < threshold){
    stop_robot(sonar6, pulse6);
}
//pulse7 = getSonar(sonar7);
// if(pulse7 < threshold){

```

```

    // stop_robot(sonar7, pulse7);
    // }
    pulse8 = getSonar(sonar8);
    if(pulse8 < threshold){
        stop_robot(sonar8, pulse8);
    }
    pulse9 = getSonar(sonar9);
    if(pulse9 < threshold){
        stop_robot(sonar9, pulse9);
    }
    break;
*/
case 4:
    // Stopped
    Serial.println("Case 4");
    break;

default:
    Serial.println("Nothing");

}

if (((distance1 < threshold) || (distance3 < threshold) || (distance4 < threshold) || (distance6 < threshold) ||
(distance7 < threshold) || (distance9 < threshold) || (distance10 < threshold) || (distance12 < threshold)) &&
((distance1 > 0) && (distance3 > 0) && (distance4 > 0) && (distance6 > 0) && (distance7 > 0) &&
(distance9 > 0) && (distance10 > 0) && (distance12 > 0)) {
    digitalWrite(InterruptPin, ALERT);
}
}

///          Get Sonar Function          ///
/* Programmer passes sonar number into this function
to return sonar reading. i.e. getSonar(sonar3); */

int getSonar(int sonar) {
    int pulse;
    for(int i = 0 ; i < arraysize; i++){
        pulse = pulseIn(sonar, HIGH);
        delay(48);
        rangevalue[i] = pulse/147;
    }
    isort(rangevalue, arraysize);
    pulse = mode(rangevalue, arraysize);
    return pulse;
}

////////// end Sonar Function //////////

///          Stop Robot Function          ///
/* If any sonar detects a reading of less than the
variable "threshold" it will call this function.

```

Starts by setting the Interrupt Pin High to notify the Mega. Then it spins in a while loop, continuously reading the sonar that detected the obstacle. Once the obstacle is removed the loop ends and the function removes the alert to the Mega by driving the interrupt pin low. \*/

```

void stop_robot(uint8_t sonar, uint8_t obstacle){
  ALERT = HIGH;
  data_to_send[0] = obstacle;
  data_to_send[1] = sonar;
  digitalWrite(InterruptPin, ALERT);
  while(obstacle < threshold){
    data_to_send[0] = obstacle;
    data_to_send[1] = sonar;
    Serial.print("obstacle: ");
    Serial.print(obstacle);
    Serial.print(" sonar: ");
    Serial.print(sonar); // Debug Block
    Serial.print(" ALERT: ");
    Serial.println(ALERT);
    Serial.print("Data_to_send0: ");
    Serial.println(data_to_send[0]);
    Serial.print("Data_to_send1: ");
    Serial.println(data_to_send[1]);
    delay(200);
    obstacle = getSonar(sonar);
  }
  ALERT = LOW;
  digitalWrite(InterruptPin, ALERT);
  //data_to_send[0] = 25;
  //data_to_send[1] = 25;
}

// function that executes whenever data is requested from master
// this function is registered as an event, see setup()
void requestEvent()
{
  //Wire.write(data_to_send, 2); // respond with message of 2 bytes
  // as expected by master
  Wire.write(rangevalue, 12); // Send out all 12 sonar measurements
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
  kinectVal = Wire.read(); // receive byte as an integer
  // Serial.println(kinectVal); // print the integer
}

// isort is used to sort the array of read values from a sonar //
void isort(uint8_t *a, int n){

```

```

for (int i = 1; i < n; ++i){
    int j = a[i];
    int k;
    for (k = i - 1; (k >= 0) && (j < a[k]); k--){
        a[k + 1] = a[k];
    }
    a[k + 1] = j;
}
}
//////////////////////////////// End isort //////////////////////////////////

// Mode function, returns the mode or median. //
int mode(uint8_t *x,int n){
    int i = 0;
    int count = 0;
    int maxCount = 0;
    int mode = 0;
    int bimodal;
    int prevCount = 0;
    while(i<(n-1)){
        prevCount=count;
        count=0;
        while(x[i] == x[i+1]){
            count++;
            i++;
        }
        if(count>prevCount&count>maxCount){
            mode=x[i];
            maxCount=count;
            bimodal=0;
        }
        if(count==0){
            i++;
        }
        if(count==maxCount){ //If the dataset has 2 or more modes.
            bimodal=1;
        }
        if(mode==0||bimodal==1){//Return the median if there is no mode.
            mode=x[(n/2)];
        }
        return mode;
    }
}
//////////////////////////////// end mode //////////////////////////////////

void scan() {
    if (kinectVal != 4) { // if not 'stop'
        /*!pulse1 = getSonar(sonar1);
        if(pulse1 < threshold){
            stop_robot(sonar1, pulse1);
            //Serial.println("Sonar 1");
        }!*/
    }
}

```

```

/*delay(200);
pulse7 = getSonar(sonar7);
if(pulse7 < threshold){
    stop_robot(sonar7, pulse7);
    //Serial.println("Sonar 7");
}*/
/*!delay(200);
pulse3 = getSonar(sonar3);
if(pulse3 < threshold){
    stop_robot(sonar3, pulse3);
    //Serial.println("Sonar 3");
}!*/
/*delay(200);
pulse9 = getSonar(sonar9);
if(pulse9 < threshold){
    stop_robot(sonar9, pulse9);
    //Serial.println("Sonar 9");
}*/
/*delay(200);
pulse12 = getSonar(sonar12);
if(pulse12 < threshold){
    stop_robot(sonar12, pulse12);
    //Serial.println("Sonar 12");
}*/
/*delay(200);
pulse6 = getSonar(sonar6);
if(pulse6 < threshold){
    stop_robot(sonar6, pulse6);
    //Serial.println("Sonar 6");
}*/
delay(200);
pulse10 = getSonar(sonar10);
if(pulse10 < threshold){
    stop_robot(sonar10, pulse10);
    //Serial.println("Sonar 10");
}
/*delay(200);
pulse4 = getSonar(sonar4);
if(pulse4 < threshold){
    stop_robot(sonar4, pulse4);
    //Serial.println("Sonar 4");
}*/
delay(200);
pulse2 = getSonar(sonar2);
if(pulse2 < threshold){
    stop_robot(sonar2, pulse2);
    //Serial.println("Sonar 2");
}
/*delay(200);
pulse8 = getSonar(sonar8);
if(pulse8 < threshold){
    stop_robot(sonar8, pulse8);
    //Serial.println("Sonar 8");
}

```



```
    */  
    delay(200);  
    pulse11 = getSonar(sonar11);  
    if(pulse11 < threshold){  
        stop_robot(sonar11, pulse11);  
        //Serial.println("Sonar 11");  
    }  
    /*delay(200);  
    pulse5 = getSonar(sonar5);  
    if(pulse5 < threshold){  
        stop_robot(sonar5, pulse5);  
        //Serial.println("Sonar 5");  
    }*/  
    delay(200);  
}  
}
```

## Arduino Mega\_1 Code

This code has different control inputs. For instance, if “w” is written in the serial monitor, the robot will move forward and if “f” the robot will start doing the wall following task. This code is made to receive control commands and make the robot move in different direction or achieve different tasks. Additionally, this code can receive commands to list some information about the robot state for example; if “e” printed in serial monitor, the program will return the encoder information (the same thing happened for LRF, Sonars and other states if the related letter is printed). The same principle is used to receive commands from MRPT. The MRPT sends messages and it will be interpreted to commands on the other side (Arduino Mega 1). The code of the Arduino Mega 1 is listed below:

```
/* This program accepts one of the following commands from a host program and performs the action shown
   on the PSU MCECS bot, and is intended for use with the MRPT navigation application:
   'l' - returns a 5pt LRF scan, 1 byte for each range in cm, 5 bytes total
   'r' - returns odometry (x,y,phi), 2 bytes for each coordinate in MU, MU, and degrees, 6 bytes total
   'w' - move forward
   'a' - rotate left
   's' - move backward
   'd' - rotate right
   'x' - stop moving
   This is intended to run on an Arduino Mega 2560 controlling two Basic Micro Roboclaw motor controllers
   (with two motors and two encoders each), as well as a parallax LRF mounted to a HiTec servo motor. There
   are also bumper switches programmed to send stop commands to the motors when activated. See the
   MCECS
   bot wiki for (a lot) more information. https://projects.cecs.pdx.edu/projects/roboticsclub-mcecsbot/wiki

   March 2013 by Mitch Barton
   V1.0
   */
#include <Wire.h>
#include <Servo.h>
// #include <SoftwareSerial.h> don't include because we are using BMserial and there's a conflict
#include <BMSerial.h>
#include <RoboClaw.h>

int left_led = 3; //led's to indicate which bumpers is triggered.
int right_led = 4;
```

```

int front_led = 6;
int back_led = 5;

int FLCorner = 39;
int FRCorner = 41;
int BLCorner = 43;
int BRCorner = 45;

int bumper_left = 22; //button on pin
int bumper_right = 24;
int bumper_front = 26;
int bumper_back = 28;
// opcode which determines case to execute
char pos;

// outgoing final LRF range measurements
byte lrf_data[5];
byte lrf_data_buf[5];

// positions for this particular servo with this particular mounting
// based on center
int center = 100;
int pos1 = center + 20;
int pos2 = center + 10;
int pos3 = center - 10;
int pos4 = center - 20;
int right90 = 170;
int left90 = 30;

// address for motor controller
#define address 0x80

// constants for PID on motor controllers
#define Kp 0x00010000
#define Ki 0x00008000
#define Kd 0x00004000
#define qpps 44000
//#define STOP 4
#define STOP 6
#define FORWARD 2
#define BACKWARD 3
#define RIGHT 0
#define LEFT 1
#define ST_RIGHT 4
#define ST_LEFT 5
#define arraysize 12

// variables for transformation from encoders to odometry (x, y, phi)
int32_t phi = 0;
int32_t phi1 = 0;
int32_t D1 = 0;
int32_t x1 = 0;

```

```

int32_t y1 = 0;
int32_t x = 0;
int32_t y = 0;
uint8_t S1 = 0, S2 = 0, S3 = 0, S4 = 0, S5 = 0, S6 = 0, S7 = 0, S8 = 0, S9 = 0, S10 = 0, S11 = 0, S12 = 0;

// the width of the wheels is 17" which converts to 0.4427 map units,
// but we've also added the scale factor for converting to map units E-8,
// as well as converting from radians to degrees
const int32_t w = 44270000*PI/180;

// assign Rx Tx serial comm for the motor controllers to pins 5 and 6
/* IMPORTANT: Not all pins can be used. Usable pins are: Pins: 10, 11, 12, 13, 50, 51, 52, 53, 62, 63, 64,
65, 66, 67, 68, 69
Using pins other than these would cause not being able to read the encoder
See this thread for more info: http://forums.basicmicro.net/robo-claw-f504/can-send-commands-to-roboclaw-but-can-t-receive-data-t9885.html
*/

// create servo object
Servo myservo;

BMSerial terminalSerial(0,1); // TerminalSerial is the debug serial windows when 0,1 is selected
RoboClaw roboclaw(50,52); // 5,6 represent the pins on the arduino that the motor controller is
connected to
//RoboClaw roboclaw(46,48); // 5,6 represent the pins on the arduino that the motor controller is
connected to
// constants for motor control

uint8_t MotorSpeed = 0; // Holds the current motorspeed of the robot, initialized at 5
uint8_t TopMotorSpeed = 30; // Top speed is actually TopMotorSpeed + 1
uint8_t NoMotorSpeed = 0; // variable to name a speed of zero
uint8_t increment = 3;//5; // Rate of acceleration. Make sure it is a multiple of TopMotorSpeed
boolean stopmoving = false; // Tells whether the robot has been asked to stop moving
boolean doneAccelerating;
boolean first_iteration = false; // As the robot accelerates it only needs to send the Uno (sonar_controller)
// when it initiates a new movement. Otherwise the controller would be bogged
// down with unnecessary communication.

boolean was_interrupt = false; // If a sonar detects and object that is too close it drives a pin high. This
// variable stores the value of that interrupt pin (currently enacted as an
// interrupt. The base code simply polls the "interrupt" pin).

int threshold = 15; // Threshold. Holds the distance at which a sonar will generate an "interrupt"
// Threshold must be the same value on the Uno(sonar_controller).

int available_movements = 6; // This is the number of movements available to the robot. Currently,
// there is only forward, backward, rotate left, and rotate right. If you add
// any movement increment this by one. It is used to make sure nothing happens
// if incorrect input values are received.

int sonar_controller = 2;
int new_movement = STOP; // Holds the command value from any input device telling what movement
to make.

```

```

int old_movement = STOP;      // Holds the last value of new_movement. Used to transition out of an old
movement                      // and into a new movement.

int update;                   // Variable to hold an intermediate value. If robot is accelerating/decelerating this
// is used to compare against new/old movements.

int interruptPin = 2;         // Pin used to signal an obstacle has been detected by sonars. Logic High
int interruptLED = 7;

//void (*function[4])(uint8_t) = {&rotateRight, &rotateLeft, &goForward, &goBackward}; // An array of
pointers to movement functions.
void (*function[6])(uint8_t) = {&rotateRight, &rotateLeft, &goForward, &goBackward, &strafeRight,
&strafeLeft}; // An array of pointers to movement functions.

int avoid_count = 0;         // how many turns it did to avoid/clear the obstacle
int avoid_direction = 6;    // which direction did it turn to avoid/clear the obstacle
int AVOID_DELAY = 1200;    // arbitrary time delay to turn
uint8_t AVOID_SPEED = 20;
uint8_t AVOID_SPEED_0 = 20;
uint8_t AVOID_SPEED_1 = 13;

// The variables in the following three lines are used in wall_following()
boolean DontMoveLeft = false;
boolean DontMoveRight = false;
uint8_t obstacle1 = 0, obstacle2 = 0, sonar_number1, sonar_number2;

uint8_t sonar_number, obstacle; // used in detected_obstacle()
//uint8_t sonar1,sonar2,sonar3,sonar4,sonar5,sonar6,sonar7,sonar8,sonar9,sonar10,sonar11,sonar12;
uint8_t sonar[arraysize] = {0,0,0,0,0,0,0,0,0,0,0,0};
uint8_t sonardata[6] = {0,0,0,0,0,0};

boolean DEBUG = false;
boolean statusRequest;
boolean wallFollowing;

boolean encoder_hack = false; // false = update x, true = update y

void setup()
{
    // open PC/arduino serial port at 9600 baud
    Serial.begin(9600);

    // open the software serial port to the LRF at 9600 baud
    Serial1.begin(9600);

    // open the software serial port for the roboclaw
    roboclaw.begin(2400);

    Wire.begin();

    pinMode(left_led, OUTPUT); //led output

```

```

pinMode(right_led, OUTPUT); //led output
pinMode(front_led, OUTPUT); //led output
pinMode(back_led, OUTPUT); //led output

pinMode(FLCorner, OUTPUT); //led output
pinMode(FRCorner, OUTPUT); //led output
pinMode(BLCorner, OUTPUT); //led output
pinMode(BRCorner, OUTPUT); //led output

pinMode(bumper_left, INPUT); //button input
pinMode(bumper_right, INPUT); //button input
pinMode(bumper_front, INPUT); //button input
pinMode(bumper_back, INPUT); //button input

digitalWrite(bumper_left,HIGH);//pull this input on.
digitalWrite(bumper_right,HIGH);//pull this input on.
digitalWrite(bumper_front,HIGH);//pull this input on.
digitalWrite(bumper_back,HIGH);//pull this input on.

// attaches the servo on pin 9 to the servo object
myservo.attach(8);

// initialize the LRF
delay(2000);
Serial1.write('U');
delay(100);
Serial1.read();

// set PID constants for motor controller
roboclaw.SetM1Constants(address,Kd,Kp,Ki,qpps);
roboclaw.SetM2Constants(address,Kd,Kp,Ki,qpps);
roboclaw.SetM1Constants(0x81,Kd,Kp,Ki,qpps);
roboclaw.SetM2Constants(0x81,Kd,Kp,Ki,qpps);

pinMode(interruptLED, OUTPUT);

// setup bumper interrupt pins
/*pinMode(2, INPUT_PULLUP);
pinMode(3, INPUT_PULLUP);
attachInterrupt(0, bumper_2, FALLING);
attachInterrupt(1, bumper_3, FALLING);*/

POST();

doneAccelerating = true;
statusRequest = false;
wallFollowing = false;
}

void loop() {

readSerial();

```

```

if (pos != 'f')readInterruptx();
//doMove();
if (doMove() && statusRequest) {
    returnAllData();
    statusRequest = false;
}
delay(150);
}

void readSerial() {

    int temp_cmd = -1;

    // if we get a valid byte, write it to our local variable
    if (Serial.available() > 0) {
        delay(150);
        //pos = Serial.read();
        //Serial.print("Key: ");
        //Serial.println(pos);
        int prev_movement = new_movement;
        new_movement = Serial.read();

        if(new_movement > 40) {
            temp_cmd = keyboardDebug(new_movement);
            if (temp_cmd > -1) { // if temp_cmd is -1, it's not movement related
                new_movement = temp_cmd;
            }
            else {
                new_movement = prev_movement;
            }
        }
    }
}

int keyboardDebug(int pos) {

    int debug_pos = -1; // value=-1 for non-movement commands
    int wf = 0;
    // depending what that byte is, perform an action
    switch (pos) {
        case 'r':
            // LRF single read
            //simpleread();
            complicatedread();
            delay(150);
            debug_pos = -1;
            break;

        case 'm':
            lrfScan1();
            debug_pos = -1;
            break;
    }
}

```

```

// l is the agreed character to start the standard 5pt LRF reading
case 'l':
  //lrfScan2();

  getSonarMrpt();
  debugPrintInt(sonardata[0]);
  debugPrint(",");
  debugPrintInt(sonardata[1]);
  debugPrint(",");
  debugPrintInt(sonardata[2]);
  debugPrint(",");
  debugPrintInt(sonardata[3]);
  debugPrint(",");
  debugPrintlnInt(sonardata[4]);

  debug_pos = -1;
  Serial.write('!');
  delay(200);
  Serial.write(sonardata,6);
  break;

case 'e':
  readEncoder();
  debug_pos = -1;
  break;

// rotate right command
case 'd':
  //debug_pos = RIGHT;
  debug_pos = FORWARD;
  encoder_hack = false;
  /*new_movement = 0;
  stopmoving = false;

  // motion smoothing, return to old motion and slow down, then go to new motion
  if(stopmoving == false && kinectVal < 5){
    kinectVal = Stop_old_movement();
    if(stopmoving == false && kinectVal < 5){
      kinectVal = Start_new_movement();
    }
  }
  */

  break;

// rotate left command
case 'a':
  debug_pos = LEFT;
  /*kinectVal = 1;
  stopmoving = false;

  // motion smoothing, return to old motion and slow down, then go to new motion
  if(stopmoving == false && kinectVal < 5){
    kinectVal = Stop_old_movement();

```



```

    if(stopmoving == false && kinectVal < 5){
        kinectVal = Start_new_movement();
    }
}*/

break;

// stop command
case 'x':
    debug_pos = STOP;
    /*kinectVal = 4;
    stopmoving = true;
    kinectVal = Stop_old_movement();

    // motion smoothing, return to old motion and slow down, then go to new motion
    if(stopmoving == false && kinectVal < 5){
        kinectVal = Stop_old_movement();
        if(stopmoving == false && kinectVal < 5){
            kinectVal = Start_new_movement();
        }
    }*/

break;

// forward command
case 'w':
    debug_pos = FORWARD;
    encoder_hack = true;
    /*kinectVal = 2;
    stopmoving = false;

    // motion smoothing, return to old motion and slow down, then go to new motion
    if(stopmoving == false && kinectVal < 5){
        kinectVal = Stop_old_movement();
        if(stopmoving == false && kinectVal < 5){
            kinectVal = Start_new_movement();
        }
    }*/

break;

// backward command
case 's':
    debug_pos = BACKWARD;
    /*kinectVal = 3;
    stopmoving = false;

    // motion smoothing, return to old motion and slow down, then go to new motion
    if(stopmoving == false && kinectVal < 5){
        kinectVal = Stop_old_movement();
        if(stopmoving == false && kinectVal < 5){
            kinectVal = Start_new_movement();
        }
    }

```

```

    */

    break;

case 'z':
    debug_pos = ST_LEFT;
    break;

case 'c':
    debug_pos = ST_RIGHT;
    break;

case 'f':
    debug_pos = wall_following();
    break;

case 'y':
    debug_pos = -1;
    statusRequest = true;
    //returnAllData();
    break;

case '?':

    if (wallFollowing) wf=1;

    Serial.write(wf);
    break;

case 'p':
    debug_pos = -1;
    if (DEBUG) {
        debugPrintln("Debug off");
        DEBUG = false;
    } else {
        DEBUG = true;
        debugPrintln("Debug on");
    }
    break;

case 48: // number 0 (zero)
    //getSonarData();
    Wire.requestFrom(sonar_controller, 12);

    delay(100);
    debugPrint("Bytes available: ");
    debugPrintlnInt(Wire.available());

    if (Wire.available() == arraysize) {
        for( int i=0; i < arraysize; i++ ) {
            sonar[i] = Wire.read();
            debugPrintInt(sonar[i]);
            debugPrint(", ");
        }
    }

```

```

    }
    Serial.println("");
}

break;

// right now the only other case is an error so we return 0
default:
lrf_data[0] = 0;
lrf_data[1] = 0;
lrf_data[2] = 0;
lrf_data[3] = 0;
lrf_data[4] = 0;

//Serial.write(lrf_data,5);
debug_pos = -1;
break;
}

return debug_pos;
}

void getSonarData() {
//delay(200);
boolean data_ok = false;

while(!data_ok) { // Keep requesting sonar data from UNO until no reading is 0
Wire.requestFrom(sonar_controller, arraysize);

delay(100);
//debugPrint("Bytes available: ");
//debugPrintlnInt(Wire.available());

if (Wire.available() == arraysize) {
for( int i=0; i < arraysize; i++ ) {
sonar[i] = Wire.read();
//if (sonar[i] == 0) {
// debugPrintln("reading sonar failed. Repeating ...");
// break; // If reading 0, retry requesting from UNO
//}
//Serial.print(sonar[i]);
debugPrintlnInt(sonar[i]);
debugPrint(",");
//Serial.print(", ");
if (i==arraysize-1) data_ok = true;
}
//Serial.println("");
debugPrintln("");
}
}
}

void getSonarMrpt() {

```

```

getSonarData();
//byte sonardata[5] = {0,0,0,0,0}
sonardata[0] = sonar[2];
sonardata[1] = sonar[0];
sonardata[2] = (sonar[0] + sonar[11])/2;
sonardata[3] = sonar[11];
sonardata[4] = sonar[9];

}

void getSonars_LED(){
getSonarData();
//Serial.println("getting sonars dude!");
if((sonar[0]<10 && sonar[0] > 0)|| (sonar[1]<10 && sonar[1] > 0) || (sonar[2]<10 && sonar[2] > 0)){
digitalWrite(FLCorner,HIGH);
debugPrintln("Front Left Corner LED is on!");}
else if (sonar[0]>=10 && sonar[1]>=10 && sonar[2]>=10){
digitalWrite(FLCorner,LOW);
debugPrintln("Front Left Corner LED is off!");}

if((sonar[3]<10 && sonar[3] > 0) || (sonar[4]<10 && sonar[4] > 0) || (sonar[5]<10 && sonar[5] > 0)){
digitalWrite(BLCorner,HIGH);
debugPrintln("Back Left Corner LED is on!");} //else digitalWrite(BLCorner,LOW);
else if (sonar[3]>=10 && sonar[4]>=10 && sonar[5]>=10){
digitalWrite(BLCorner,LOW);
debugPrintln("Back Left Corner LED is off!");}

if((sonar[6]<10 && sonar[6] > 0) || (sonar[7]<10 && sonar[7] > 0) || (sonar[8]<10 && sonar[8] > 0)){
digitalWrite(BRCorner,HIGH); //else digitalWrite(BRCorner,LOW);
debugPrintln("Back Right Corner LED is on!");}
else if (sonar[6]>=10 && sonar[7]>=10 && sonar[8]>=10){
digitalWrite(BRCorner,LOW);
debugPrintln("Back Right Corner LED is off!");}

if((sonar[9]<10 && sonar[9] > 0) || (sonar[10]<10 && sonar[10] > 0) || (sonar[11]<10 && sonar[11] > 0)){
digitalWrite(FRCorner,HIGH);// else digitalWrite(FLCorner,LOW);
debugPrintln("Front Right Corner LED is on!");}
else if (sonar[9]>=10 && sonar[10]>=10 && sonar[11]>=10){
digitalWrite(FRCorner,LOW);
debugPrintln("Front Right Corner LED is off!");}

int wall_following() {
//Serial.println();
boolean Fr_Obs = false;
boolean Left_Obs = false;
boolean Right_Obs = false;
//DontMoveLeft = false;
//DontMoveRight = false;
//int right_dis, left_dis;
int in_Left = 0, in_Right = 0, in_Front = 0, in_Back = 0;
do {
wallFollowing = true;

```

```

getSonarData();
S1 = sonar[0]; //Front left sonar
S3 = sonar[2]; //right sonar
S4 = sonar[3]; //right sonar
S9 = sonar[8]; //left sonar
S10 = sonar[9]; //left sonar
S12 = sonar[11]; //Front right sonar

debugPrintInt(S1);
debugPrint(" ");
debugPrintInt(S3);
debugPrint(" ");
debugPrintInt(S4);
debugPrint(" ");
debugPrintInt(S9);
debugPrint(" ");
debugPrintInt(S10);
debugPrint(" ");
debugPrintlnInt(S12);

getSonars_LED();

delay(100);
in_Left = digitalRead(bumper_left);
in_Right = digitalRead(bumper_right);
in_Front = digitalRead(bumper_front);
in_Back = digitalRead(bumper_back);

// IF no bumper is hit, then continue with wall following
if (in_Left == HIGH && in_Right == HIGH && in_Front == HIGH && in_Back == HIGH) {
  digitalWrite(left_led,LOW);
  digitalWrite(right_led,LOW);
  digitalWrite(front_led,LOW);
  digitalWrite(back_led,LOW);

  debugPrint("\nRight distance: ");
  debugPrintlnInt(S10); //Print right distance
  debugPrint("Left distance: ");
  debugPrintlnInt(S3); // print left distance

  if ((S1 < 10 && S1 > 0) || (S12 < 10 && S12 > 0)) Fr_Obs = true; else Fr_Obs = false;
  debugPrintln("checking front obstacle");

  if ((S3 < S10) || (S3 == S10)) {
    if (Fr_Obs) {
      new_movement = ST_RIGHT;
      Left_Obs = true;
      Right_Obs = false;
      //avoid(new_movement);
      debugPrintln("(Following Left) obstacle and moving right");
      // goto start;
    } else {

```

```

if (Left_Obs){
  new_movement = FORWARD;
  Left_Obs = false;
  //avoid(new_movement);
  debugPrintln("(Following Left) avoiding obstacle and moving forward");
  // goto start;
} else {
//No_ObsL:
  if (S3 < 10 && S3 > 0) {

    // check bumper
    /*in = digitalRead(bumper_left);
    // Keep stopping while bumper is on
    while (in == LOW) {
      goStop();
      in = digitalRead(bumper_left);
    }*/
    //if (left_dis < 33) goto start;
    new_movement = ST_RIGHT;
    /*getSonarData();
    //S1 = sonar[0];
    /*S3 = sonar[2];
    //S4 = sonar[3];
    //S9 = sonar[8];
    /*S10 = sonar[9];
    //S12 = sonar[11];
    //avoid(new_movement);
    debugPrintln("(Following Left) less than 10 and moving right");

    //goto No_ObsL;
  } else if (S3 >= 10) {
    if (S3 > 20){
      // Read from bumper
      /*in = digitalRead(bumper_left);
      // Keep stopping while bumper is on
      while (in == LOW) {
        goStop();
        in = digitalRead(bumper_left);
      }*/
      new_movement = ST_LEFT;
      /*getSonarData();
      //S1 = sonar[0];
      /*S3 = sonar[2];
      // S4 = sonar[3];
      // S9 = sonar[8];
      /*S10 = sonar[9];
      // S12 = sonar[11];
      //avoid(new_movement);
      debugPrintln("(Following Left) larger than 20 and moving left");

      //goto No_ObsL;
    } else {
      new_movement = FORWARD;

```

```

        debugPrintln("(Following Left) Forward");
        //avoid(new_movement);
        // goto start;
    }
}
}
} else {
if (Fr_Obs){
    new_movement = ST_LEFT;
    Right_Obs = true;
    Left_Obs = false;
    debugPrintln("(Following right) obstacle and moving left");
    //avoid(new_movement);
    // goto start;
} else {
    if (Right_Obs) {
        new_movement = FORWARD;
        Right_Obs = false;
        debugPrintln("(Following right) avoiding obstacle and moving forward");
        //avoid(new_movement);
        // goto start;
    } else {
//No_ObsR:
        if (S10 < 10 && S10 > 0) {
            //if (right_dis < 33) goto start;
            new_movement = ST_LEFT;
            /*getSonarData();
            // S1 = sonar[0];
            /*S3 = sonar[2];
            //S4 = sonar[3];
            //S9 = sonar[8];
            /*S10 = sonar[9];
            //S12 = sonar[11];
            debugPrintln("(Following Right) less than 10 and moving left");
            //avoid(new_movement);

            // Check bumper
            /*
            in = digitalRead(bumper_left);
            // Keep stopping while bumper is on
            while (in == LOW) {
                goStop();
                in = digitalRead(bumper_left);
            }*/
            //goto No_ObsR;
        } else if (S10 >= 10) {
            if (S10 > 20){
                new_movement = ST_RIGHT;
                /*getSonarData();
                //S1 = sonar[0];
                /*S3 = sonar[2];
                // S4 = sonar[3];

```

```

        // S9 = sonar[8];
        /*S10 = sonar[9];
        //S12 = sonar[11];
        debugPrintln("(Following Right) larger than 20 and moving right");
        //avoid(new_movement);

        // Read from bumper
        /* in = digitalRead(bumper_left);
        // Keep stopping while bumper is on
        while (in == LOW) {
            goStop();
            in = digitalRead(bumper_left);
        }*/

        //goto No_ObsR;
        }
        else {
            new_movement = FORWARD;
            debugPrintln("(Following right) Forward");
            //avoid(new_movement);
            // goto start;
        }
    }
}
//avoid(new_movement);
}

debugPrint("Fr_Obs: ");
debugPrintlnInt(Fr_Obs);
debugPrint("Right_Obs: ");
debugPrintlnInt(Right_Obs);
debugPrint("Left_Obs: ");
debugPrintlnInt(Left_Obs);

} else {
    goStop(); // Stop the motors and indicate which bumpers has been tregered using the serial moniter and
led's.
    if (in_Left == LOW) {
        digitalWrite(left_led,HIGH);
        debugPrintln(": LEFT BUMPER");
    }
    if (in_Right == LOW) {
        digitalWrite(right_led,HIGH);
        debugPrintln(": RIGHT BUMPER");
    }
    if (in_Front == LOW) {
        digitalWrite(front_led,HIGH);
        debugPrintln(": FRONT BUMPER");
    }
    if (in_Back == LOW) {
        digitalWrite(back_led,HIGH);
        debugPrintln(": BACK BUMPER");
    }
}

```



```

    }

    }
    if (Serial.available() > 0) {
        char serialin = Serial.read();
        if (serialin == 'x') break;
        else if (serialin == 'y') statusRequest = true;
    }
    if (doMove() && statusRequest) {
        returnAllData();
        statusRequest = false;
    }
} while(true);

Serial.read();
debugPrintln("Done wall following!");
new_movement = STOP;
wallFollowing = false;
avoid(new_movement);
return -1;

}
void readInterrupt() {
    getSonarData();
    //getSonars_LED();
    if (((sonar[0] < threshold) || (sonar[2] < threshold) || (sonar[3] < threshold) || (sonar[5] < threshold) ||
(sonar[6] < threshold) || (sonar[8] < threshold) || (sonar[9] < threshold) || (sonar[11] < threshold)) &&
((sonar[0] > 0) && (sonar[2] > 0) && (sonar[3] > 0) && (sonar[5] > 0) && (sonar[6] > 0) && (sonar[8] >
0) && (sonar[9] > 0) && (sonar[11] > 0)))) {
        digitalWrite(interruptLED, HIGH);
        detected_obstacle();
    }
}

void readInterruptx() {
    was_interrupt = digitalRead(interruptPin); // Poll interrupt pin from sonar controller.
    if(was_interrupt == true){
        //Serial.println("Interrupt true.");
        digitalWrite(interruptLED, HIGH);
        if(new_movement == FORWARD || new_movement == BACKWARD) {
            detected_obstacle(); // jump to stop routine for detected obstacle
        }
        was_interrupt = false;
    }
    else {
        digitalWrite(interruptLED, LOW);
        //Serial.println("No interrupt.");
        /*if (avoid_count > 0) {
            //goForward(15);
            //delay(AVOID_DELAY);
            Serial.print("Avoid count: ");
            Serial.println(avoid_count);
            Serial.print("Direction: ");

```

```

    if(avoid_direction == 0) {
      Serial.println("RIGHT");
    }
    else { Serial.println("LEFT");
    }
    returnToPath();
  }
  if (avoid_count == 0) {
    avoid_direction = 4;
  }*/
}
}

//////////////////////////////////// Read Interrupt End //////////////////////////////////////

//////////////////////////////////// Begin LRF and Encoder subroutines //////////////////////////////////////

void lrfScan1()
{
  myservo.write(pos1);
  delay(1000);
  simpleread();
  //complicatedread();

  myservo.write(pos2);
  delay(20);
  simpleread();
  //complicatedread();
  //delay(1000);
  myservo.write(center);
  delay(20);
  simpleread();
  //complicatedread();
  //delay(1000);
  myservo.write(pos3);
  delay(20);
  simpleread();
  //complicatedread();
  //delay(1000);
  myservo.write(pos4);
  delay(20);
  simpleread();
  //complicatedread();
  //delay(1000);
  centerPos();
}

void lrfScan2()
{
  debugPrintln("\nLRF Scan2");
  // The ! is the arduino acknowledge for the LRF reading
  Serial.write('!');
}

```

```

// Move servo, start LRF reading, store in outgoing array, repeat 5 times
myservo.write(pos1);
delay(120);
Serial1.write('R');
//lrf_data[0] = dataread();
lrf_data[0] = complicatedread();
lrf_data_buf[0] = lrf_data[0];
//delay(1500);

myservo.write(pos2);
delay(120);
Serial1.write('R');
//lrf_data[1] = dataread();
lrf_data[1] = complicatedread();
lrf_data_buf[1] = lrf_data[1];
//delay(1500);

myservo.write(center);
delay(120);
Serial1.write('R');
//lrf_data[2] = dataread();
lrf_data[2] = complicatedread();
lrf_data_buf[2] = lrf_data[2];
//delay(1500);

myservo.write(pos3);
delay(120);
Serial1.write('R');
//lrf_data[3] = dataread();
lrf_data[3] = complicatedread();
lrf_data_buf[3] = lrf_data[3];
//delay(1500);

myservo.write(pos4);
delay(120);
Serial1.write('R');
//lrf_data[4] = dataread();
lrf_data[4] = complicatedread();
lrf_data_buf[4] = lrf_data[4];
//delay(1500);

// send out the data array
centerPos();
Serial.write(lrf_data,5);
//delay(1000);
}

void readEncoder()
{
// status byte for encoders
uint8_t status;

```

```

// bit to determine if encoder values are valid
bool valid;

// character buffer for opcode
//char pos;

// local variables for caclulating odometry
int32_t enc1;
int32_t enc2;

int32_t Dleft;
int32_t Dright;

// angle in radians for the trig functions
float phi1fl;
//Serial.println();
//Serial.flush();
while (Serial.available() > 0) {
  Serial.read();
}
// predetermined acknowledge char which motherboard requires to accept odometry data
Serial.write('*');

// read encoder, and set status and valid bits
enc1= roboclaw.ReadEncM1(address, &status, &valid);
if(valid) {

  // prevents underflow, since we are only looking for the relative change in enc cts and resetting it every
  loop,
  // if the enc subtracts from 0, it will end up overflowing the 16bit variable as well as being incorrect for
  our
  // incremental case, for example an enc value reported as 4294967290 will be converted to -5 which is the
  true
  // relative change from 0 as we expect it. This could possibly be improved by using the status byte which
  will
  // indicate an over or underflow.

  if (enc1 > (2^16)) {
    enc1 = enc1 - (2^32) + 1;
  }

  // 1470cts/rev, 2.0944ft/rev, 3.2ft/map unit leads to 0.00044524 map units/encoder ct, or 44524 map units
  E-8/enc ct
  Dright = enc1*44524;

  // debug prints
  /*Serial.print("Encoder1: ");
  Serial.print(enc1,DEC);
  Serial.print(", ");
  Serial.print(Dright,DEC);
  Serial.println(" ");*/
  debugPrint("Encoder1: ");

```

```

    debugPrintInt(enc1);
    debugPrint(" ");
    debugPrintlnInt(Dright);

}

// read encoder 2, and set status and valid bits
enc2= roboclaw.ReadEncM2(address, &status, &valid);
if(valid) {

    // prevents underflow, since we are only looking for the relative change in enc cts and resetting it every
    loop,
    // if the enc subtracts from 0, it will end up overflowing the 16bit variable as well as being incorrect for
    our
    // incremental case, for example an enc value reported as 4294967290 will be converted to -5 which is the
    true
    // relative change from 0 as we expect it.
    if (enc2 > (2^16)) {
        enc2 = enc2 - (2^32) + 1;
    }

    // 1470cts/rev, 2.0944ft/rev, 3.2ft/map unit leads to 0.00044524 map units/encoder ct, or 44524 map units
    E-8/enc ct
    Dleft = enc2*44524;

    // debug prints
    /*debugPrint("Encoder2:");
    debugPrint((char *)enc2,DEC);
    debugPrint(" ");
    debugPrint((char *)Dleft,DEC);
    Serial.println(" ");*/
    debugPrint("Encoder2:");
    debugPrintInt(enc2);
    debugPrint(" ");
    debugPrintlnInt(Dleft);

}

// the following equations are taken from simreal.com/content/Odometry, and assume a two wheel 'tank-
type' robot, which
// for MCECS bot means that all motion is assumed to have equal motion from the two left wheels and two
right wheels. It
// is certainly capable of other motion, but these equations MUST be rewritten to account for those motions.
They are
// derived by using the law of sines and assuming the distance is a straight line.

// iterative calculation which accumulates the angular change of the robot from left and right encoder cts,
where w is
// the width of the two wheels
phi1 = phi + (Dleft/w) - (Dright/w);

// keeps angle wrapping around the circle either positively or negatively
if (phi1 < 0)

```

```

{
  phi1 += 360;
}
else if (phi1 >= 360)
{
  phi1 -= 360;
}

// new change in distance, approximating curved path as a straight line. The max speed of the robot and the
// sample period
// for the encoders determines the amount of error, that is the faster the robot moves and the larger the period
// the larger
// the approximation error. We determined that our max speed is sufficiently low and our sample period is
// sufficiently
// large such that the error is negligible. If the speed or sample period are altered that may no longer be the
// case!!!
D1 = (Dleft/2) + (Dright/2);

// convert angle to radians and cast as float for trig functions
phi1fl = float(phi1*PI/180);

// iterative calculation which accumulates the x, y position of the robot based on these previous values
// obtained from
// encoder cts
//x1 = x + D1*sin(phi1fl);
//y1 = y + D1*cos(phi1fl);
if (!encoder_hack) x1 = x + D1;
else y1 = y + D1;

// iterate the transform variables
x=x1;
y=y1;
phi=phi1;

x1 /= 1000000;
y1 /= 1000000;
//D1 /= 1000000;

// debug prints
debugPrint("Distance: ");
//Serial.println(D1);
debugPrintlnInt(D1);
debugPrint("X coordinate: ");
//Serial.println(x1,DEC);
debugPrintlnInt(x1);
debugPrint("Y coordinate: ");
//Serial.println(y1,DEC);
debugPrintlnInt(y1);
debugPrint("Angle in radians: ");

if (DEBUG) { // Float number - can't use debugPrint* - function doesn't accept float
  Serial.print(phi1fl);
}

```

```

debugPrint("Angle in degrees: ");
//Serial.println(phi1,DEC);
debugPrintlnInt(phi1);

byte package[6];
package[0] = x1 & 255; // x1
if (x1 < 0) {
  package[1] = ((x1 >> 8) & 255) | 128;
}
else
{
  package[1] = ((x1 >> 8) & 255) & 127;
}
package[2] = y1 & 255; // y1
if (y1 < 0)
{
  package[3] = ((y1 >> 8) & 255) | 128;
}
else
{
  package[3] = ((y1 >> 8) & 255) & 127;
}

package[4] = phi1 & 255;
package[5] = (phi1 >> 8) & 255;

Serial.write(package,6);
//Serial.flush();
while (Serial.available() > 0) {
  Serial.read();
}

// reset the encoders so that we are receiving incremental change in encoder cts
roboclaw.ResetEncoders(address);
delay(1000);
}

void centerPos()
{
  myservo.write(center);
}

// A simple method to read from the LRF
void simpleread() {
  debugPrintln("Reading LRF ...");
  char readlrf;
  char readbuffer[4];
  long result;
  boolean gotresult = false;
  // Sometimes serial buffer is not empty - clear it out first or you'll get erroneous data
  debugPrintln("Flushing ... ");
  while(Serial1.available() > 0) {

```

```

    Serial1.read();
    debugPrint(".");
}
debugPrint("done.\n");

delay(10);
Serial1.write('R');
int count = 0;
boolean lt15 = false;
while(Serial1.available() < 15) {
    if (!lt15) {
        debugPrintln("Serial1.available < 15 ...");
        lt15 = true;
    }
    //delay(100);
    count = count + 1;
    if (count > 20) {
        //delay(150);
        Serial1.write('R');
        delay(50);
        count = 0;
    }
}
//delay(1000);
debugPrint("Serial1.available() = ");
debugPrintlnInt(Serial1.available());
while (Serial1.available() > 0) {

    //delay(150);
    readlrf = Serial1.read();
    delay(10);
    if (readlrf == ':') {
        //debugPrintln(":");
        readlrf = Serial1.read();
    }

    debugPrint((char*)readlrf);
}

//Serial1.flush();
debugPrintln("Done!");
}

// A more complicated method to read from LRF, the output is formatted in long type
long complicatedread() {
    //debugPrintln("Reading LRF ...");
    char readlrf;
    char readbuffer[4];
    long result;
    boolean gotresult = false;
    //Serial.println();
    // Sometimes serial buffer is not empty - clear it out first or you'll get erroneous data
    //debugPrintln("Flushing ... ");

```



```

while(Serial1.available() > 0) {
  Serial1.read();
  //debugPrint(".");
}
//debugPrintln("done.");

delay(10);
Serial1.write('R');
int count = 0;
//debugPrintln("Scanning ... \n");
while(Serial1.available() < 15) {
  //Serial.println("Serial1.available < 15");
  //Serial.print(".");
  delay(10);

  // keep reading until Serial1.available() >= 15
  count = count + 1;
  // if already checked 20 times (arbitrary number) and still failing, send the command again
  if (count > 50) {
    //delay(150);
    Serial1.write('R');
    delay(20);
    count = 0;
  }
}
if (Serial1.available() == 16) {
  //debugPrintln("LRF scan failed.\n");
  return 0;
}
//Serial.write("\n");
//delay(1000);
//Serial.write("Serial1.available() = ");
//Serial.println(Serial1.available());
while (Serial1.available() > 0) {

  //delay(150);
  readlrf = Serial1.read();
  delay(10);
  if (readlrf == ':') {
    //Serial.println(':');
    readlrf = Serial1.read();
  }
  //Serial.print("readlrf: ");
  //Serial.println((char)readlrf);

  // Format of response from lrf: "D = 0000 mm"
  if ((readlrf == 'D') && !gotresult) {
    Serial1.read(); // space
    Serial1.read(); // '='
    Serial1.read(); // space
    readbuffer[0] = Serial1.read();
    readbuffer[1] = Serial1.read();
    readbuffer[2] = Serial1.read();
  }
}

```

```

    readbuffer[3] = Serial1.read();
    // in mm: result = (readbuffer[0]-'0')*1000 + (readbuffer[1]-'0')*100 + (readbuffer[2]-'0')*10 +
(readbuffer[3]-'0');
    result = (readbuffer[0]-'0')*100 + (readbuffer[1]-'0')*10 + (readbuffer[2]-'0');
    debugPrint("Distance: ");
    debugPrintInt(result);
    debugPrint(" cm\n");
    gotresult = true;
    break;
    //delay(200);
}

//Serial.print(readlrf);
}
//Serial.write("Flushing ... ");
while(Serial1.available()) {
    Serial1.read();
}
//Serial1.flush();
//Serial.println("Done!");
return result;
}

// dataread will read the bytes from the LRF and buffer them, then assemble them into a value in cm
long dataread() {
    long result;
    char range[15];
    //Serial.println("datareading...");
    debugPrintln("reading from LRF...");
    while (Serial1.available() < 15) {
        debugPrint(".");
    }
    for (int i = 0; i < 15; i++) {
        range[i] = Serial1.read();
        delay(50);
    }

    //round the mm value to an integer cm value
    if (range[10] < 6) {
        result = (range[7]-'0')*100 + (range[8]-'0')*10 + (range[9]-'0');
    }
    else
    {
        result = (range[7]-'0')*100 + (range[8]-'0')*10 + (range[9]-'0') + 1;
    }

    // if the measurement is outside the valid range of the LRF return a 0
    if (result > 240 || result < 15) {
        result = 0;
    }
}

```

```

// flush buffer
for (int j = 0; j < 15; j++) {
    range[j] = 0;
}

return result;
}
//////////////////////////////////// POST start //////////////////////////////////////

// This code turns the robot right then left to show the robot is in working order and able to move

void POST() {
    debugPrint("POST Test Start");
    rotateRight(10);
    delay(500);
    rotateLeft(10);
    delay(500);
    goStop();
    delay(500);
    debugPrint("POST Test End");
}

//////////////////////////////////// POST end //////////////////////////////////////

//////////////////////////////////// Do Move start //////////////////////////////////////

boolean doMove() {
    if(new_movement == STOP && old_movement != STOP) {
        //debugPrintln("new_movement==STOP && old_movement!=STOP");
        while(MotorSpeed > 10) {
            MotorSpeed = MotorSpeed - increment;
            function[old_movement](MotorSpeed);
            delay(1);
        }
        MotorSpeed = 0;
        goStop();
        old_movement = new_movement;
        return true;
    }
    else if(new_movement != old_movement) {
        //debugPrintln("new_movement!=old_movement");
        while(MotorSpeed > 10) {
            MotorSpeed = MotorSpeed - increment;
            function[old_movement](MotorSpeed);
            delay(1);
        }
        function[new_movement](MotorSpeed);
        old_movement = new_movement;
        return true;
    }
    else if(MotorSpeed < TopMotorSpeed && old_movement != STOP) {
        //debugPrintln("MotorSpeed<TopMotorSpeed && old_movement!=STOP");
        MotorSpeed = MotorSpeed + increment;
    }
}

```

```

    if(MotorSpeed > TopMotorSpeed){
        MotorSpeed = TopMotorSpeed;
        return true;
    }
    function[new_movement](MotorSpeed);
    old_movement = new_movement;
    return false;
}

// DEBUG OUTPUT
/*Serial.print("Old_movement= ");
Serial.print(old_movement);
Serial.print(", new_movement= ");
Serial.println(new_movement);*/
/*debugPrint("Old_movement= ");
debugPrint((char*)old_movement);
debugPrint(", new_movement= ");
debugPrintlnInt(new_movement);*/

old_movement = new_movement;
}

//////////////////////      Do Move End      ////////////////////////

//////////////////////      Stop Old Movement Function      ////////////////////////
/*
When we jump into this loop we set update to the value of new_movement. This
variable allows us to make sure that if we are sent a new movement during this
loop we have a variable to compare to. We then decelerate from the old movement
until we reach a speed of 0. At which point we update the old_movement to the
new_movement.

If we enter this loop and we have been told that to stop (stopmoving = true)
then new_movement becomes equal to old_movement.

We continually check to make sure we are not receiving new movement data.
If we do receive new movement data we jump out of Stop_old_movement into the
main loop.

We always send back new_movement to the main loop in case it has changed.
*/
int Stop_old_movement(){
    update = new_movement;
    while( new_movement != old_movement && MotorSpeed > NoMotorSpeed){
        debugPrintln("stopping old movement ...");
        MotorSpeed = MotorSpeed - increment;
        first_iteration = false;
        function[old_movement](MotorSpeed);

        if (MotorSpeed == NoMotorSpeed && stopmoving == true){
            new_movement = old_movement;
            debugPrintln("updated kinect value from STOP");
        }
    }
}

```

```

    else if (MotorSpeed == NoMotorSpeed){
    old_movement = new_movement;
    debugPrintln("updated kinect value from STOP");
    }

    if(Serial.available()){
    update = Serial.read();
    if (update != new_movement && update != 10){
    new_movement = update;
    //Serial.println(update);
    //Serial.println("Interrupted Stop Movement");
    debugPrintlnInt(update);
    debugPrintln("Interrupted Stop Movement");
    break;
    }
    }
}
return new_movement;
}
//////////      End stop Old Movement Function      //////////

//////////      Start New Movement Function      //////////
/*

```

Similar to the stop function the start continually checks for new movement input. It accelerates up to the TopMotorSpeed and leaves the loop. first\_iteration is set to false after the first call of a movement function so that the Mega does not continue sending movement data to the sonar controller when the movement isn't changing.

```

*/
int Start_new_movement(){
    update = new_movement;
    for( MotorSpeed; MotorSpeed <= TopMotorSpeed; MotorSpeed = MotorSpeed + increment){
    function[new_movement](MotorSpeed);
    first_iteration = false;
    if(MotorSpeed == TopMotorSpeed){
        old_movement = new_movement;
        debugPrintln("updated kinect from START");
    }
}

    if(Serial.available()){
    update = Serial.read();
    if (update != new_movement && update != 10){
    old_movement = new_movement;
    new_movement = update;
    //Serial.println(update);
    //Serial.println("Interrupted Start Movement");
    debugPrintlnInt(update);
    debugPrintln("Interrupted Start Movement");
    break;
    }
    }
}

```

```

return new_movement;
}

// routine to slow down old motion for motor commands
/*int Stop_old_movement(){
update = kinectVal;
//Serial.print("pressed: ");
//Serial.println(kinectVal);
while( kinectVal != oldKinectVal && MotorSpeed > NoMotorSpeed){
MotorSpeed = MotorSpeed - increment;
function[oldKinectVal](MotorSpeed);

if (MotorSpeed == NoMotorSpeed && stopmoving == true){
kinectVal = oldKinectVal;
//Serial.println("updated kinect value from STOP");
}
if (MotorSpeed == NoMotorSpeed){
oldKinectVal = kinectVal;
//Serial.println("updated kinect value from STOP");
}

if(Serial.available()){
update = Serial.read();
if (update != kinectVal){
kinectVal = update;
//Serial.println("Interrupted Stop Movement");
break;
}
}
}
return kinectVal;
}*/

// routine to smoothly start new motion
/*int Start_new_movement(){
update = new_movement;
//Serial.print("pressed: ");
//Serial.println(new_movement);
for( MotorSpeed; MotorSpeed <= TopMotorSpeed; MotorSpeed = MotorSpeed + increment){

function[new_movement](MotorSpeed);

if(MotorSpeed == TopMotorSpeed){
old_movement = new_movement;
//Serial.println("updated kinect from START");
}

if(Serial.available()){
update = Serial.read();
if (update != new_movement && update != 10){
old_movement = new_movement;
new_movement = update;
//Serial.println("Interrupted Start Movement");
}
}
}
}*/

```

```

        break;
    }
}
}
return new_movement;
}*/

void rotateRight(uint8_t MotorSpeed){
    debugPrint("Rotate Right: ");
    debugPrintlnInt(MotorSpeed);
    roboclaw.BackwardM1(0x80, MotorSpeed);
    roboclaw.ForwardM2(0x80, MotorSpeed);
    roboclaw.BackwardM1(0x81, MotorSpeed);
    roboclaw.ForwardM2(0x81, MotorSpeed);
}

void rotateLeft(uint8_t MotorSpeed){
    debugPrint("Rotate Left: ");
    debugPrintlnInt(MotorSpeed);
    roboclaw.ForwardM1(0x80, MotorSpeed);
    roboclaw.BackwardM2(0x80, MotorSpeed);
    roboclaw.ForwardM1(0x81, MotorSpeed);
    roboclaw.BackwardM2(0x81, MotorSpeed);
}

void strafeLeft(uint8_t MotorSpeed){
    debugPrint("Strafe Left: ");
    debugPrintlnInt(MotorSpeed);
    roboclaw.ForwardM1(0x80, MotorSpeed);
    roboclaw.BackwardM2(0x80, MotorSpeed);
    roboclaw.BackwardM1(0x81, MotorSpeed);
    roboclaw.ForwardM2(0x81, MotorSpeed);
}

void strafeRight(uint8_t MotorSpeed){
    debugPrint("Strafe Right: ");
    debugPrintlnInt(MotorSpeed);
    roboclaw.BackwardM1(0x80, MotorSpeed);
    roboclaw.ForwardM2(0x80, MotorSpeed);
    roboclaw.ForwardM1(0x81, MotorSpeed);
    roboclaw.BackwardM2(0x81, MotorSpeed);
}

void goForward(uint8_t MotorSpeed){
    debugPrint("Forward: ");
    debugPrintlnInt(MotorSpeed);
    roboclaw.ForwardM1(0x80, MotorSpeed);
    roboclaw.ForwardM2(0x80, MotorSpeed);
    roboclaw.ForwardM1(0x81, MotorSpeed);
    roboclaw.ForwardM2(0x81, MotorSpeed);
}

```

```

Wire.beginTransmission(sonar_controller);
Wire.write(new_movement);
Wire.endTransmission();
}

void goBackward(uint8_t MotorSpeed){
  debugPrint("Backward: ");
  debugPrintlnInt(MotorSpeed);
  roboclaw.BackwardM1(0x80, MotorSpeed);
  roboclaw.BackwardM2(0x80, MotorSpeed);
  roboclaw.BackwardM1(0x81, MotorSpeed);
  roboclaw.BackwardM2(0x81, MotorSpeed);
  Wire.beginTransmission(sonar_controller);
  Wire.write(new_movement);
  Wire.endTransmission();
}

void goForwardDiag(uint8_t MotorSpeedRight, uint8_t MotorSpeedLeft){
  debugPrint("Diagonal Forward: ");
  debugPrintlnInt(MotorSpeed);
  roboclaw.ForwardM1(0x80, MotorSpeedRight);
  roboclaw.ForwardM2(0x80, MotorSpeedLeft);
  roboclaw.ForwardM1(0x81, MotorSpeedRight);
  roboclaw.ForwardM2(0x81, MotorSpeedLeft);
}

void goBackwardDiag(uint8_t MotorSpeedRight, uint8_t MotorSpeedLeft){
  debugPrint("Diagonal Reverse: ");
  debugPrintlnInt(MotorSpeed);
  roboclaw.BackwardM1(0x80, MotorSpeedRight);
  roboclaw.BackwardM2(0x80, MotorSpeedLeft);
  roboclaw.BackwardM1(0x81, MotorSpeedRight);
  roboclaw.BackwardM2(0x81, MotorSpeedLeft);
}

void goStop(){
  debugPrint("Stopping");
  MotorSpeed = 0;
  roboclaw.ForwardM1(0x80, MotorSpeed);
  roboclaw.ForwardM2(0x80, MotorSpeed);
  roboclaw.ForwardM1(0x81, MotorSpeed);
  roboclaw.ForwardM2(0x81, MotorSpeed);
}

/*
void displayspeed(void)
{
  uint8_t status;
  bool valid;

  uint32_t enc1 = roboclaw.ReadEncM1(0x80, &status, &valid);
  if(valid){

```



```

Serial.print("Encoder1:");
Serial.print(enc1,DEC);
Serial.print(" ");
Serial.print(status,HEX);
Serial.print(" ");
}
uint32_t enc2 = roboclaw.ReadEncM1(0x81, &status, &valid);
if(valid){
Serial.print("Encoder2:");
Serial.print(enc2,DEC);
Serial.print(" ");
Serial.print(status,HEX);
Serial.print(" ");
}
uint32_t speed1 = roboclaw.ReadSpeedM1(0x80, &status, &valid);
if(valid){
Serial.print("Speed1:");
Serial.print(speed1,DEC);
Serial.print(" ");
}
uint32_t speed2 = roboclaw.ReadSpeedM1(0x81, &status, &valid);
if(valid){
Serial.print("Speed2:");
Serial.print(speed2,DEC);
Serial.print(" ");
}
Serial.println();
}
*/

void bumper_2()
{
roboclaw.ForwardM1(0x80,0);
roboclaw.ForwardM2(0x80,0);
roboclaw.ForwardM1(0x81,0);
roboclaw.ForwardM2(0x81,0);
//Serial.println("b2");
//delay(25);
}

void bumper_3()
{
roboclaw.ForwardM1(0x80,0);
roboclaw.ForwardM2(0x80,0);
roboclaw.ForwardM1(0x81,0);
roboclaw.ForwardM2(0x81,0);
//Serial.println("b3");
//delay(25);
}

// Detected Obstacle Function //
/*

```

This function runs when the sonar controller sets the Mega pin 2 HIGH. Indicated that an obstacle is in the way of the robot. The Uno will send which sonar detected the obstacle and how far the obstacle is from that sonar. If the obstacle is less than 10 inches away it will perform an emergency stop where the robot stops immediately. If 10inches < obstacle < 15inches the robot will decelerate. This function will get stuck in the while loop if the obstacle remains, thereby preventing the robot from moving until the obstacle has been removed.

```

*/
void detected_obstacle() {
  //uint8_t sonar_number, obstacle;
  int interrupted_movement;
  interrupted_movement = new_movement;

  debugPrint("Obstacle Detected: ");

  if(old_movement == BACKWARD) {
    //new_movement = FORWARD;
    avoid(FORWARD);
  }
  else {
    //new_movement = BACKWARD;
    avoid(BACKWARD);
    avoid(RIGHT);
  }

  avoid(STOP);
  getSonarData();
  getSonars_LED();
}

void detected_obstaclex(){
  //uint8_t sonar_number, obstacle;
  int interrupted_movement;
  interrupted_movement = new_movement;

  debugPrint("Obstacle Detected: ");

  if(old_movement == BACKWARD) {
    //new_movement = FORWARD;
    avoid(FORWARD);
  }
  else {
    //new_movement = BACKWARD;
    avoid(BACKWARD);
  }

  avoid(STOP);

  //char sonar_number, obstacle;

```

```

Wire.requestFrom(sonar_controller, 2);
//int i = 0;
delay(100);
debugPrint("Bytes available: ");
debugPrintlnInt(Wire.available());

delay(100);
obstacle = Wire.read();
delay(100);
sonar_number = Wire.read();
debugPrint("\nobstacle: ");
debugPrintInt(obstacle);
debugPrint(" sonar number: ");
debugPrintlnInt(sonar_number);

///////// evasive manuever method //////////
decide(obstacle, sonar_number);

///////// End of evasive manuever method //////////

        new_movement = interrupted_movement;
        debugPrintln("end detected_obstacle()");
}

// End Detected Obstacle Function //

// Decide what to do / where to move when there's an obstacle
void decide(int obstacle, int sonar_number) {
// Sonars:
// front left: 1,2,3
// back left: 4,5,6
// back right: 7,8,9
// front right: 10,11,12
debugPrint("Obstacle: ");
debugPrintlnInt(obstacle);
debugPrint("Sonar number: ");
debugPrintlnInt(sonar_number);
debugPrint("Trying to decide what to do...");
if ((sonar_number == 12) || (sonar_number == 11)) {
    debugPrintln("Go Left");
    avoid(LEFT);
} else if ((sonar_number == 7) || (sonar_number == 6) || (sonar_number == 5)) {
    debugPrintln("Go Right");
    avoid(RIGHT);
} else if ((sonar_number == 2) || (sonar_number == 3) || (sonar_number == 4)) {
    debugPrintln("Go Left");
    avoid(LEFT);
} else if ((sonar_number == 8) || (sonar_number == 9) || (sonar_number == 10)) {
    debugPrintln("Go Right");
    avoid(RIGHT);
} else {
    avoid(STOP);
}
}

```

```

}

void avoid(int direction) {
  long now = millis();
  int maxtime = 801;
  int mintime = 400;
  new_movement = direction;
  while (millis() - now < random(mintime,maxtime)) {
    doMove();
  }
}

// Mathias' code to read arduino data from Processing/Kinect program
void returnAllData() {
  byte interrupted = 0;
  if (was_interrupt) {
    interrupted = 1;
  } else {
    interrupted = 0;
  }
  byte stoppedmoving = 0;
  if (stopmoving) {
    stoppedmoving = 1;
  } else {
    stoppedmoving = 0;
  }
  getSonarData();

  // byte 33 = '!'
  byte dataArray[22] = { 35, interrupted, stoppedmoving, (byte)new_movement,
    lrf_data_buf[0], lrf_data_buf[1], lrf_data_buf[2],
    lrf_data_buf[3], lrf_data_buf[4],
    (byte)sonar[0], (byte)sonar[1],(byte)sonar[2],(byte)sonar[3],
    (byte)sonar[4],(byte)sonar[5],(byte)sonar[6],(byte)sonar[7],
    (byte)sonar[8],(byte)sonar[9],(byte)sonar[10],(byte)sonar[11], 36}; // need to add orientation
  and all sonar data

  Serial.write(dataArray, 22);
  /*Serial.println((char)33);
  Serial.println(interrupted);
  Serial.println(stoppedmoving);
  Serial.println(new_movement);
  Serial.println(lrf_data_buf[0]);
  Serial.println(lrf_data_buf[1]);
  Serial.println(lrf_data_buf[2]);
  Serial.println(lrf_data_buf[3]);
  Serial.println(lrf_data_buf[4]);
  Serial.println(obstacle);
  Serial.println(sonar_number);
  Serial.println((char)37);*/
}

void debugPrint(char* toPrint) {

```

```
if(DEBUG) {  
    Serial.print(toPrint);  
}  
}  
  
void debugPrintln(char* toPrint) {  
    if(DEBUG) {  
        Serial.println(toPrint);  
    }  
}  
  
void debugPrintInt(int toPrint) {  
    if(DEBUG) {  
        Serial.print(toPrint);  
    }  
}  
  
void debugPrintlnInt(int toPrint) {  
    if(DEBUG) {  
        Serial.println(toPrint);  
    }  
}
```

## MRPT Code

```
/*
 * guidebot Navigation, spring 2012
 *
=====
=
 * Tested platform: Linux + mrpt lib, guidebbot (ActivMediaRobot) base + Kinect
 * For initial MRPT library and Kinect setup please see our project report.
 * additional files: guidebotNavConf.ini, floorplan (map), run script(optional)
 *
=====
=
 * implementing:
 * - main thread: command prompt for
 * + robot status
 * + manual drive
 * + path planning and driving robot to target
 * - display thread: 3D display of current robot movement, also shows sensor
 * readings(kinect and sonar) and particles distribution
 * - pdf update thread: execute montecarlo localization using particles filter
 * based on the a current pair of action-observation.
 * - kinect observation grabbing thread
 * - wall detect thread
 *
=====
=
 * Current status and issues
 * - path planning and 3d display works
 * - driving is smooth, however does not work well with wall detect.
 * - localization works in principle however needed some tuning. Particles are
 * calculated in background and updated to our display ONLY. We have NOT update
 * current robot location (robot odometry, where robot thinks it is at) with the
 * most likely location (calculated from particle filter). We think there is an
 * issue with the changeOdometry() function from MRPT. A temporary fix for this
 * (odometryOffset) is implemented, but not thoroughly tested.
 */
#include <mrpt/hwdrivers.h>
#include <mrpt/maps.h>
#include <mrpt/system/filesystem.h>
#include <mrpt/bayes/CKalmanFilterCapable.h>

#include <mrpt/base.h>
#include <mrpt/slam.h>
#include <mrpt/gui.h>
#include <mrpt/utis/CImage.h>
#include <mrpt/slam/COccupancyGridMap2D.h>
#include <mrpt/hwdrivers/CActivMediaRobotBase.h>
#include <iostream>

//Includes for arduino serial communication
#include <stdio.h>
#include <stdlib.h>
```

```

#include <sys/ioctl.h>
#include <fcntl.h>
#include <termios.h>
#include <fstream>
#include <iostream>
#include <string>
#include <unistd.h>

//includes for network communication
#include <sys/socket.h>
#include <arpa/inet.h>

using namespace mrpt;
using namespace mrpt::hwdrivers;
using namespace mrpt::utils;
using namespace mrpt::math;
using namespace mrpt::slam;
using namespace mrpt::poses;
using namespace mrpt::gui;
using namespace mrpt::bayes;
using namespace mrpt::random;
using namespace std;

#define CONFIG_FILE_NAME "guidebotNavConf.ini"

#define BEARING_SENSOR_NOISE_STD      DEG2RAD(15.0f)
#define RANGE_SENSOR_NOISE_STD      0.3f
#define DELTA_TIME                    0.1f

#define VEHICLE_INITIAL_X            0.0f
#define VEHICLE_INITIAL_Y            0.0f
#define VEHICLE_INITIAL_V            1.0f
#define VEHICLE_INITIAL_W            DEG2RAD(20.0f)

#define TRANSITION_MODEL_STD_XY 0.03f
#define TRANSITION_MODEL_STD_VXY 0.20f

/*****
*****/
/*
                NAVIGATION PARAMS
                */
/*****
*****/
/* DEFAULT PARAM, SEE "guidebotNavConf.ini" */
static string TTY_PORT    = "/dev/ttyACM0";
static string COM_PORT    = "COM4";
static int BAUD_RATE      = 9600;

static float ROBOT_RADIUS = 0.30f;
static float MIN_STEP     = 0.40f;

static double TURN_THRESHOLD = M_PI/90; /* stop condition while turning */

```

```

static double FORWARD_THRESHOLD = 0.09f; /* stop condition while forward */
static double SHARP_TURN = M_PI/5; /* when robot should stop and turn */
static int POLL_INTERVAL = 100; /* delay interval between sensor readings */
static int POLL_INTERVAL_DIV = 5;

static double ANGULAR_SPEED = 0.3; /* angular velocity */
static double LINEAR_SPEED = 0.2; /* linear velocity */
static double SMALL_NUMBER = 0.001; /* a useful number for comparing purpose */
static int ANGULAR_SPEED_DIV = 5;
static int LINEAR_SPEED_DIV = 2;

static string MAP_FILE = "FAB-LL-Central-200px.png";
static float MAP_RESOLUTION = 0.048768f;
static int X_CENTRAL_PIXEL = -1; /* start location, (-1,-1) means center of map */
static int Y_CENTRAL_PIXEL = -1;

static CMonteCarloLocalization2D pdf;
static float kinectMinTruncateDistance = 0.5;

static char* LRF_PORT_NAME = "/dev/ttyACM0";
static char* ODO_PORT_NAME = "/dev/ttyACM0";
static char* MOTOR_PORT_NAME = "/dev/ttyACM0";
static int ODO_READ_MIN = 7;
static int LRF_READ_MIN = 6;
#define PORT "80" // the port client will be connecting to

#define MAXDATASIZE 100 // max number of bytes we can get at once

#define LOCAL_HOST "127.0.0.1"

/* our threads 's sharing resources */
struct TThreadRobotParam
{
    TThreadRobotParam() : quit(false), pushed_key(0), Hz(0) { }

    mrpt::synch::CThreadSafeVariable<bool> quit;
    mrpt::synch::CThreadSafeVariable<bool> stop;
    mrpt::synch::CThreadSafeVariable<int> pushed_key;
    volatile double Hz;
    mrpt::synch::CThreadSafeVariable<COccupancyGridMap2D> * gridmap; //
    mrpt::synch::CThreadSafeVariable<CPose2D> currentOdo;
    mrpt::synch::CThreadSafeVariable<CPose2D> currentKF;
    mrpt::synch::CThreadSafeVariable<CPose2D> currentSonar;
    mrpt::synch::CThreadSafeVariable<CPose2D> targetOdo;

    mrpt::synch::CThreadSafeVariable<CObservation2DRangeScanPtr> new_obs; // RGB+D
    (+3D points)

```



```

mrpt::synch::CThreadSafeVariable<CObservationIMUPtr>    new_obs_imu; // Accelerometers
//mrpt::synch::CThreadSafeVariable<CActivMediaRobotBase> * robot; // robot info
//mrpt::synch::CThreadSafeVariable<CPose2D>            new_pose; // robot info

//mrpt::synch::CThreadSafeVariable<CActionCollectionPtr> actions;
//mrpt::synch::CThreadSafeVariable<CSensoryFramePtr>    observations;
//mrpt::synch::CThreadSafeVariable<CMonteCarloLocalization2D> pdf;
mrpt::synch::CThreadSafeVariable<CObjectPtr>          pdf;
mrpt::synch::CThreadSafeVariable<CParticleFilter>     PF;
mrpt::synch::CThreadSafeVariable<bool>
displayNewPdf;

mrpt::synch::CThreadSafeVariable<std::deque<poses::TPoint2D> > thePath;
mrpt::synch::CThreadSafeVariable<bool>
displayNewPath;

mrpt::synch::CThreadSafeVariable<deque<TSegment3D> >          sonars;
mrpt::synch::CThreadSafeVariable<bool>
displayNewSonars;

mrpt::synch::CThreadSafeVariable<bool>
displayClearOldPdf;
mrpt::synch::CThreadSafeVariable<bool>
displayClearOldSonar;
mrpt::synch::CThreadSafeVariable<bool>
displayClearOldPath;

mrpt::synch::CThreadSafeVariable<bool>
pdfResetDeterministic;

mrpt::synch::CThreadSafeVariable<bool>
leftObstacle;
mrpt::synch::CThreadSafeVariable<bool>
rightObstacle;
mrpt::synch::CThreadSafeVariable<bool>
centerObstacle;
mrpt::synch::CThreadSafeVariable<bool>
isMoving; //set when robot is moving, clear otherwise
mrpt::synch::CThreadSafeVariable<bool>
gettingLRF; //set when robot is using LRF, clear otherwise
mrpt::synch::CThreadSafeVariable<bool>
isTurning; //set when robot is using LRF, clear otherwise
mrpt::synch::CThreadSafeVariable<bool>
goForward;
mrpt::synch::CThreadSafeVariable<bool>
goRight;

mrpt::synch::CThreadSafeVariable<CPose2D>
odometryOffset;

mrpt::synch::CThreadSafeVariable<int>
odo_fd; //file descriptor for odometry port

```

```

    mrpt::synch::CThreadSafeVariable<int>
    lrf_fd; //file descriptor for lrf (may be same as odometry)
    mrpt::synch::CThreadSafeVariable<int>
    vel_fd;
    mrpt::synch::CThreadSafeVariable<float>          front_wall;
    mrpt::synch::CThreadSafeVariable<float>          right_wall;
    mrpt::synch::CThreadSafeVariable<float>          D1;
    //mrpt::synch::CThreadSafeVariable<CPose2D>
    pdfMean;
    //mrpt::synch::CThreadSafeVariable<CPose2D>
    pdfMostLikely;
};

float initial_front_wall = 0.0f;
float initial_right_wall = 0.0f;
bool initialize_walls = 1;

int scanTest = 0; //Testing global variable. Delete After test
/* prototypes */
double turnAngle(CActivMediaRobotBase & aRobot, double phi, TThreadRobotParam thrPar);
double turnAngle(double current_phi, double phi);
static void turn( double phi, TThreadRobotParam &p);
static int PathPlanning(std::deque<poses::TPoint2D> &thePath, CPoint2D origin, CPoint2D target);
static void smoothDrive(CActivMediaRobotBase & aRobot, deque<poses::TPoint2D> aPath,
TThreadRobotParam & thrPar);
static CObservation2DRangeScan* getKinect2DScan(const TThreadRobotParam & TP,
CObservation3DRangeScanPtr & lastObs);
void thread_LRF(TThreadRobotParam &p);
void thread_update_pdf(TThreadRobotParam &p);
void thread_display(TThreadRobotParam &p);
void createCObservationRange( CObservationRange &obs );
void displaySonars(TThreadRobotParam &p, CObservationRange &sonars);
static void displayFollowPath(TThreadRobotParam &p, deque<poses::TPoint2D> thePath);
double CObservationRangeLikelihood(COccupancyGridMap2D & map, CPose2D & pose,
CObservationRange & obs);
void computePdfLikelihoodValues(COccupancyGridMap2D & map, CMonteCarloLocalization2D & pdf,
CObservationRange & obs);
void adjustCObservationRangeSonarPose( CObservationRange &obs );
void thread_wall_detect(TThreadRobotParam &p);
void fixOdometry(CPose2D & pose, CPose2D offset);
int getNextObservation(CObservation2DRangeScan & out_obs, bool there_is, bool hard_error, int
fd, TThreadRobotParam &thrPar);
void getOdometry(CPose2D &out_odom, int odo_fd, TThreadRobotParam &thrPar);
int setupArduino(char * port, int readBytes);
int clientCommunication();
int parseServerReply(char * server_reply);
void setVelocities(int linear, int angular, TThreadRobotParam &thrPar);

// -----
//           Implementation of the system models as a
// -----
class CRangeBearing :

```

```

        //public mrpt::bayes::CKalmanFilterCapable<4 /* x y vx vy*/, 2 /* range yaw */, 0
Atime */>
        // <size_t VEH_SIZE, size_t
OBS_SIZE, size_t FEAT_SIZE, size_t ACT_SIZE, size typename kftype = double>
        public mrpt::bayes::CKalmanFilterCapable<2 /* x y vx vy*/, 2 /* range yaw */, 0
Atime */>
        // <size_t VEH_SIZE, size_t
OBS_SIZE, size_t FEAT_SIZE, size_t ACT_SIZE, size typename kftype = double>
{
public:
    CRangeBearing();
    virtual ~CRangeBearing();

    //void doProcess( double DeltaTime, double observationRange, double observationBearing );
    void doProcess( double sonar12,double sonar10, double dy, double dx );

    void getState( KFVector &xkk, KFMatrix &pkk)
    {
        xkk = m_xkk;
        pkk = m_pkk;
    }

protected:

    float m_obsBearing,m_obsRange;
    float m_deltaTime;
    float m_sonar12; // latest sonar reading
    float m_sonar10;
    float m_dy; // current positon - previous position
    float m_dx;

    /** @name Virtual methods for Kalman Filter implementation
    @{
    */

    /** Must return the action vector u.
    * \param out_u The action vector which will be passed to OnTransitionModel
    */
    void OnGetAction( KFArray_ACT &out_u ) const;

    /** Implements the transition model  $\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k)$ 
    * \param in_u The vector returned by OnGetAction.
    * \param inout_x At input has  $\hat{x}_{k-1|k-1}$ , at output must have  $\hat{x}_{k|k-1}$ 
    \f$.
    * \param out_skip Set this to true if for some reason you want to skip the prediction step (to do not
    modify either the vector or the covariance). Default:false
    */
    void OnTransitionModel(
        const KFArray_ACT &in_u,
        KFArray_VEH &inout_x,
        bool &out_skipPrediction
    ) const;

```

```

/** Implements the transition Jacobian  $\frac{\partial f}{\partial x}$ 
 * \param out_F Must return the Jacobian.
 * The returned matrix must be  $N \times N$  with N being either the size of the whole state
vector or get_vehicle_size().
 */
void OnTransitionJacobian(KFMatrix_VxV &out_F) const;

/** Implements the transition noise covariance  $Q_k$ 
 * \param out_Q Must return the covariance matrix.
 * The returned matrix must be of the same size than the jacobian from OnTransitionJacobian
 */
void OnTransitionNoise(KFMatrix_VxV &out_Q) const;

/** Return the observation NOISE covariance matrix, that is, the model of the Gaussian additive
noise of the sensor.
 * \param out_R The noise covariance matrix. It might be non diagonal, but it'll usually be.
 * \note Upon call, it can be assumed that the previous contents of out_R are all zeros.
 */
void OnGetObservationNoise(KFMatrix_OxO &out_R) const;

/** This is called between the KF prediction step and the update step, and the application must return
the observations and, when applicable, the data association between these observations and the current map.
 *
 * \param out_z N vectors, each for one "observation" of length OBS_SIZE, N being the number of
"observations": how many observed landmarks for a map, or just one if not applicable.
 * \param out_data_association An empty vector or, where applicable, a vector where the i'th
element corresponds to the position of the observation in the i'th row of out_z within the system state vector
(in the range [0,getNumberOfLandmarksInTheMap()-1]), or -1 if it is a new map element and we want to
insert it at the end of this KF iteration.
 * \param in_all_predictions A vector with the prediction of ALL the landmarks in the map. Note
that, in contrast, in_S only comprises a subset of all the landmarks.
 * \param in_S The full covariance matrix of the observation predictions (i.e. the "innovation
covariance matrix"). This is a  $M \cdot O \times M \cdot O$  matrix with  $M = \text{length of "in_lm_indices_in_S"}$ .
 * \param in_lm_indices_in_S The indices of the map landmarks (range
[0,getNumberOfLandmarksInTheMap()-1]) that can be found in the matrix in_S.
 *
 * This method will be called just once for each complete KF iteration.
 * \note It is assumed that the observations are independent, i.e. there are NO cross-covariances
between them.
 */
void OnGetObservationsAndDataAssociation(
    vector_KFArray_OBS &out_z,
    mrpt::vector_int &out_data_association,
    const vector_KFArray_OBS &in_all_predictions,
    const KFMatrix &in_S,
    const vector_size_t &in_lm_indices_in_S,
    const KFMatrix_OxO &in_R
);

/** Implements the observation prediction  $h_i(x)$ .
 * \param idx_landmark_to_predict The indices of the landmarks in the map whose
predictions are expected as output. For non SLAM-like problems, this input value is undefined and the
application should just generate one observation for the given problem.

```

```

    * \param out_predictions The predicted observations.
    */
void OnObservationModel(
    const vector_size_t &idx_landmarks_to_predict,
    vector_KFArray_OBS &out_predictions
) const;

    /** Implements the observation Jacobians  $\frac{\partial h_i}{\partial x}$  and (when
    applicable)  $\frac{\partial h_i}{\partial y_i}$ .
    * \param idx_landmark_to_predict The index of the landmark in the map whose
    prediction is expected as output. For non SLAM-like problems, this will be zero and the expected output is
    for the whole state vector.
    * \param Hx The output Jacobian  $\frac{\partial h_i}{\partial x}$ .
    * \param Hy The output Jacobian  $\frac{\partial h_i}{\partial y_i}$ .
    */
void OnObservationJacobians(
    const size_t &idx_landmark_to_predict,
    KFMatrix_OxV &Hx,
    KFMatrix_OxF &Hy
) const;

    /** Computes  $A=A-B$ , which may need to be re-implemented depending on the topology
    of the individual scalar components (eg, angles).
    */
void OnSubstractObservationVectors(KFArray_OBS &A, const KFArray_OBS &B)
const;

    /** @ }
    */
};

/*****
*****/
/*                               FUNCTION IMPLEMENTATIONS                               */
/*****
*****/

/*
 * @Description
 * This thread does montecarlo localization and put the pdf into ThreadRobotParam.
 * Calculation requires a current observation-action pair, from kinect and odometry.
 * A map is also needed. Tuning parameters defined in guidebotNavConf.ini for:
 * - Motion and sensor model
 * - PDF options
 * - Particle filter options
 *
 * @param      TThreadRobotParam p (for current odometry)
 * @return     none
 */
void thread_update_pdf(TThreadRobotParam &p)
{
    CPose2D currentOdo, previousOdo;
    CConfigFile guidebotConfFile(CONFIG_FILE_NAME);

```

```

COccupancyGridMap2D      gridmap;
CParticleFilter::TParticleFilterStats PF_stats;

/* get the map first */
bool thereis;
CImage img;
float   resolution = MAP_RESOLUTION;           // size of the grid in meters
float   xCentralPixel = X_CENTRAL_PIXEL;       // x central pixel
float   yCentralPixel = Y_CENTRAL_PIXEL;       // y central pixel // Load
the gridmap:

if (!mrpt::system::fileExists(MAP_FILE))
    THROW_EXCEPTION_CUSTOM_MSG1("Map      file      '%s'      not
found",MAP_FILE.c_str());

printf("Loading gridmap...");
gridmap.loadFromBitmapFile(MAP_FILE,resolution ,xCentralPixel,yCentralPixel);
printf("Done! %f x %f m\n", gridmap.getXMax()-gridmap.getXMin(), gridmap.getYMax()-
gridmap.getYMin());

/* insert and likelihood observation options for gridmap */
COccupancyGridMap2D::TInsertionOptions gridmapOption;
COccupancyGridMap2D::TLikelihoodOptions likelihoodOption;

int ind = 0;

gridmapOption.loadFromConfigFile(                       guidebotConfFile,
"MetricMap_occupancyGrid_00_insertOpts" );
gridmapOption.loadFromConfigFile(                       guidebotConfFile,
"MetricMap_occupancyGrid_00_likelihoodOpts" );

gridmap.insertionOptions = gridmapOption;
gridmap.likelihoodOptions = likelihoodOption;

/***** MonteCarloLocalization *****/

uint64_t M = guidebotConfFile.read_uint64_t("LocalizationParams","PARTICLE_COUNT",1000,
false);
pdf = CMonteCarloLocalization2D(M);

/* PDF Options: */
TMonteCarloLocalizationParams pdfPredictionOptions;
pdfPredictionOptions.KLD_params.loadFromConfigFile( guidebotConfFile, "KLD_options");
pdf.options = pdfPredictionOptions;

pdf.options.metricMap = &gridmap;

/* PF-algorithm Options: */
CParticleFilter::TParticleFilterOptions      pfOptions;
pfOptions.loadFromConfigFile( guidebotConfFile, "PF_options" );

CParticleFilter  PF;

```

```

PF.m_options = pfOptions;

/* motion model */
CActionRobotMovement2D::TMotionModelOptions dummy_odom_params;
dummy_odom_params.modelSelection = CActionRobotMovement2D::mmGaussian;
dummy_odom_params.gaussianModel.minStdXY
guidebotConfFile.read_double("DummyOdometryParams","minStdXY",0.04);
dummy_odom_params.gaussianModel.minStdPHI
DEG2RAD(guidebotConfFile.read_double("DummyOdometryParams","minStdPHI", 2.0));

CPose2D pdfEstimation, odometryEstimation;
//CPose2D currentOdo, previousOdo;

/* get a COPY */
previousOdo.x(p.currentOdo.get().x());
previousOdo.y(p.currentOdo.get().y());
previousOdo.phi(p.currentOdo.get().phi());

/* reset all particle to a known location */
pdf.resetDeterministic(previousOdo,1000);

/* this part below uniformly distributes particles to the whole map */

// if ( !guidebotConfFile.read_bool("LocalizationParams","init_PDF_mode",false, /*Fail if
not found*/true) )
// pdf.resetUniformFreeSpace(
// &gridmap,
// 0.7f,
// M ,
// guidebotConfFile.read_float("LocalizationParams","init_PDF_min_x",0,true),
// guidebotConfFile.read_float("LocalizationParams","init_PDF_max_x",0,true),
// guidebotConfFile.read_float("LocalizationParams","init_PDF_min_y",0,true),
// guidebotConfFile.read_float("LocalizationParams","init_PDF_max_y",0,true),
//
// DEG2RAD(guidebotConfFile.read_float("LocalizationParams","init_PDF_min_phi_deg",-180)),
//
// DEG2RAD(guidebotConfFile.read_float("LocalizationParams","init_PDF_max_phi_deg",180))
// );
// else
// pdf.resetUniform(
// guidebotConfFile.read_float("LocalizationParams","init_PDF_min_x",0,true),
// guidebotConfFile.read_float("LocalizationParams","init_PDF_max_x",0,true),
// guidebotConfFile.read_float("LocalizationParams","init_PDF_min_y",0,true),
// guidebotConfFile.read_float("LocalizationParams","init_PDF_max_y",0,true),
//
// DEG2RAD(guidebotConfFile.read_float("LocalizationParams","init_PDF_min_phi_deg",-180)),
//
// DEG2RAD(guidebotConfFile.read_float("LocalizationParams","init_PDF_max_phi_deg",180)),
// M
// );

//CObservation3DRangeScanPtr last_obs;

```

```

/* repeat pdf calculation until terminated */
while(p.quit.get() == false)
{
    if ( p.pdfResetDeterministic.get() )
    {
        pdf.resetDeterministic(previousOdo,1000);
        p.pdfResetDeterministic.set(false);
    }

    /* pair of action-observation */
    CActionCollectionPtr actions;
    CSensoryFramePtr senFrame;

    CObservationPtr obsPtr;

    /* get a copy of the most current pose */
    currentOdo.x(p.currentOdo.get().x());
    currentOdo.y(p.currentOdo.get().y());
    currentOdo.phi(p.currentOdo.get().phi());

    /* it IS possible that computeFromOdometry return nothing if distance
    * between pose is tiny, adjust minStdXY if need */

    if (currentOdo.distanceTo(previousOdo) >
2*dummy_odom_params.gaussianModel.minStdXY)
    {
        /* make sure that we have new kinect reading */
        CObservation2DRangeScanPtr obs_Ptr = p.new_obs.get();
        CObservation2DRangeScan* obs_2d = obs_Ptr.pointer();

        /* make sure that we have new kinect reading */
        if (obs_2d == NULL)
            continue;
        obs_2d->truncateByDistanceAndAngle(kinectMinTruncateDistance,3);
        senFrame = CSensoryFrame::Create(); /* get a sensoryFrame Ptr */
        actions = CActionCollection::Create(); /* get an AC Ptr */
        /* insert action */
        CActionRobotMovement2DPtr dummy_odom =
CActionRobotMovement2D::Create();
        dummy_odom->computeFromOdometry(currentOdo - previousOdo,
dummy_odom_params);
        actions->insert(*dummy_odom);

        /* stamp this pose */
        previousOdo.x(currentOdo.x());
        previousOdo.y(currentOdo.y());
        previousOdo.phi(currentOdo.phi());

        /* insert observation */
        obsPtr.setFromPointerDoNotFreeAtDtor(obs_2d);
        senFrame->insert(obsPtr);

        /* add observation to our map for visualization */

```



```

CPose3D currentOdo3D(currentOdo);
obs_2d->insertObservationInto(&gridmap, &currentOdo3D);

//      cout << "----- PDF -----" << endl;
PF.executeOn(
    pdf,
    actions.pointer(),           // Action
    senFrame.pointer(),         // Obs.
    &PF_stats                    // Output statistics
);

CActionRobotMovement2DPtr    best_mov_estim    =    actions-
>getBestMovementEstimation();

    if (best_mov_estim)
    {
        odometryEstimation = odometryEstimation + best_mov_estim-
>poseChange->getMeanVal();
        //      cout << "odometryEstimation : " << odometryEstimation << endl;
    }
    pdf.getMean( pdfEstimation );
    //cout << "pdfEstimation : " << pdfEstimation << endl;
    //cout << "mostlikelyParticle: " << pdf.getMostLikelyParticle() << endl;

    p.pdf.set( pdf.duplicateGetSmartPtr() );
    p.displayNewPdf.set(true);

    //cout << "adaptive sample size : " << PF.adaptiveSampleSizepdf.size() << endl;
    //cout << "-----" << endl;

    /* end if */
    mrpt::system::sleep(POLL_INTERVAL);
    /* end while */

    /* done and save outputs */
    pdf.saveToTextFile("pdf.txt"); /* Save PDF's m_particles to a text file.*/
    //gridmap.getAsImage(img,false, true); /* Force a RGB image */
    //const std::string dest = "path_planning1.png";
    //cout << "Saving output to: " << dest << endl;
    //img.saveToFile(dest);
}

/*
 * @Description
 * Thread for grabbing: monitor kinect reading, this is required for getKinect2DScan()
 * calibration file can be used for more accurate reading.
 */
void thread_LRF(TThreadRobotParam &p)
{
    //get lrf file descriptor
    int fd = p.lrf_fd.get();
    sleep(1000);

```

```

try
{
    const std::string cfgFile = CONFIG_FILE_NAME;
    if (mrpt::system::fileExists(cfgFile))
    {
        cout << "Loading calibration from: " << cfgFile << endl;
        // Put any code needed to configure LRF from config file
    }
    else cerr << "Warning: Calibration file [" << cfgFile << "] not found -> Using default
params.\n";

    // Open:
    cout << "Calling LRF::initialize()...";
    //Put code to initialize LRF :::
    cout << "OK\n";

    CTicTac tictac;
    int nImgs = 0;
    bool there_is_obs=true, hard_error=false;
    int odometryDelayCounter = 0;

    /* monitor and update hardware status */
    while (!hard_error && !p.quit.get() )
    {
        /* Grab new observation from the camera:*/
        CObservation2DRangeScanPtr obs = CObservation2DRangeScan::Create();
        CObservationIMUPtr obs_imu = CObservationIMU::Create();
        if(!p.isMoving.get() & !p.gettingLRF.get())
        {
            getNextObservation(*obs,there_is_obs,hard_error,fd,p);
        }
        if (!hard_error && there_is_obs )
        {
            p.new_obs.set(obs);
            // p.new_obs_imu.set(obs_imu);
        }
        sleep(2500);
    }/* end while*/
}
catch(std::exception &e)
{
    cout << "Exception in grabbing thread: " << e.what() << endl;
    p.quit.set(true);
}
}

/*
 * @Description
 *This function is made to mock the getNextObservation function from the kinect
 *
 * @param      obs: pointer to a CObservation2DRangeScan object created from the LRF
 *             there_is_obs: boolean set if there is a new observation observation
 */

```

```

*          hard_error: true when an error occurs with the LRF
* @return   none
*/
int getNextObservation(CObservation2DRangeScan & out_obs, bool there_is, bool hard_error, int fd,
TThreadRobotParam &thrPar)
{
    unsigned char buf[256];
    char getCommand[1];
    int n;
    cout<<"lrf_odo "<<fd<<endl;
    cout<<thrPar.gettingLRF.get()<<"LRF VALUE"<<endl;

    thrPar.gettingLRF.set(true);
    getCommand[0]='I';
    /* Flush anything already in the serial buffer */
    cout<<thrPar.gettingLRF.get()<<"LRF VALUE"<<endl;
    fflush(fd, TCIFLUSH);
    /* read up to 128 bytes from the fd */
    write(fd,getCommand,1);
    buf[0] = 0;
    while(buf[0] != '!')
    {
        n = read(fd, buf, 7);

        sleep(100);
        printf("%i sonar bytes got read...\n", n);
        printf("Buffer has \n%s\n",buf);
    }

    sleep(150);

    printf("%i bytes got read...\n", n);
    printf("Buffer 1 contains...\n%d (%.03f)\n", buf[1], buf[1]/38.4);
    printf("Buffer 2 contains...\n%d (%.03f)\n", buf[2], buf[2]/38.4);
    printf("Buffer 3 contains...\n%d (%.03f)\n", buf[3], buf[3]/38.4);
    printf("Buffer 4 contains...\n%d (%.03f)\n", buf[4], buf[4]/38.4);
    printf("Buffer 5 contains...\n%d (%.03f)\n", buf[5], buf[5]/38.4);

    /*
    CPose2D      odo;
    double       v,w;
    int64_t      left_ticks, right_ticks;
    bool         pollLRF = true;
    cout<<"counter getting odo"<<endl;
    p.isMoving.set(true);
    while(p.gettingLRF.get());
    //{
    //    pollLRF = returnGettingLRF(thrPar);
    //    cout<<pollLRF<<endl;
    //    sleep(1000);
    */

```

```

    //}

    cout<<"OK"<<endl;
    getOdometry( odo, fd, p );
    p.isMoving.set(false);
    printf("***x = %d, y = %d, phi = %d\n",odo.x(),odo.y(),odo.phi());
    fixOdometry( odo, p.odometryOffset.get() );
    p.currentOdo.set(odo);
    printf("***x = %d, y = %d, phi = %d\n",odo.x(),odo.y(),odo.phi());
    cout << "Odometry: " << odo << " v: " << v << " w: " << RAD2DEG(w) << " left: " << left_ticks
<< " right: " << right_ticks << endl;
    */

    CPose2D curOdo = thrPar.currentOdo.get();
    CPose2D newOffset(curOdo.x(),curOdo.y(),curOdo.phi());
    out_obs.scan.clear();
    fixOdometry( newOffset, thrPar.odometryOffset.get() );
    out_obs.validRange.clear();
    out_obs.setSensorPose(thrPar.odometryOffset.get());
    out_obs.aperture = M_PI;/*120/180;
    sleep(1000);

    //for(int i = 1; i < LRF_READ_MIN; i++)
    for(int i=5; i > 0; i--)
    {
//        printf("Buffer %d contains...\n%d\n",i,buf[i]);
        if (i > 0)
        {
            //out_obs.scan.push_back(float(buf[i])/100);
            out_obs.scan.push_back(float(buf[i])/38.4); // inches converted into metric / map
unit
            out_obs.validRange.push_back(1);
            cout << "yak!";
        }
    }

    if (initialize_walls) {
run        initial_front_wall = float(buf[2])/38.4; // Set initial front wall only the first time the program

        initial_right_wall = float(buf[5])/38.4;
        initialize_walls = 0;
    }

    thrPar.front_wall.set(float(buf[2])/38.4); // Save the front sonar reading
    thrPar.right_wall.set(float(buf[5])/38.4); // Save the right sonar reading

    thrPar.gettingLRF.set(false);
    cout<<thrPar.gettingLRF.get()<<"LRF VALUE"<<endl;

    return 0;
}

/*

```

```

* @Description
* Replacement for the getOdometry function that was part of the CActivMediaRobotBase class
* @param out_odom: a pointer to the object in which the current odometry of the robot will get filled
in
*
*/
void getOdometry(CPose2D &out_odom, int odo_fd, TThreadRobotParam &thrPar)
{
    unsigned char buf[256];
    char getCommand[1];
    int n;
    CPose2D tempPose;
    short x,y,phi;
    getCommand[0]='e';
    cout<<"In getting ODO"<<endl;
    /* Flush anything already in the serial buffer */
    tcflush(odo_fd, TCIFLUSH);
    /* read up to 128 bytes from the fd */
    write(odo_fd,getCommand,1);
    cout<<"written"<<endl;
    cout<<"odo_fd " <<odo_fd<<endl;
    buf[0] = 0;
    while(buf[0] != '*')
    {
        cout<<"reading"<<endl;
        n = read(odo_fd, buf, 7);

        sleep(100);
        printf("%i odometry bytes got read...\n", n);
        printf("Buffer has \n%s\n",buf);
    }

    sleep(150);

    //read data in
    n = read(fd, buf,6);

    // printf("Buffer has \n%s\n",buf);
    // printf("%i bytes got read...\n", n);
    // printf("Buffer 1 contains...\n%d\n", buf[0]);
    // printf("Buffer 2 contains...\n%d\n", buf[1]);
    // printf("Buffer 3 contains...\n%d\n", buf[2]);
    // printf("Buffer 4 contains...\n%d\n", buf[3]);
    // printf("Buffer 5 contains...\n%d\n", buf[4]);
    // printf("Buffer 5 contains...\n%d\n", buf[5]);
    // printf("Buffer 5 contains...\n%d\n", buf[6]);
    cout<<"read"<<endl;
    x = ((buf[2] & 255)<< 8) | (buf[1] & 255);
    y = ((buf[4] & 255) << 8) | (buf[3] & 255);
    phi = ((buf[6] & 255) << 8) | (buf[5] & 255);

    //flip phi over x axis

```

```

//      if(phi != 0)
//      {
//          phi = 360 - phi;
//      }
printf("x = %d, y = %d, phi = %d\n",x,y,phi);
tempPose = thrPar.currentOdo.get();

thrPar.D1.set(float(x)/100.0);
//x = x + tempPose.x();
//y = y + tempPose.y();
//phi = phi + tempPose.phi();

out_odom.x(float(x)*0.707/100.0);
out_odom.y(float(y)*0.707/100.0);
//out_odom.phi((float(phi)+90)*M_PI/180.0);

/*if (thrPar.goRight.get() && !thrPar.goForward.get()) {
cout << "UPDATING X*****\n";
    out_odom.x(thrPar.currentOdo.get().x() + float(x)/100.0);
    out_odom.y(thrPar.currentOdo.get().y());
    out_odom.phi(90*M_PI/180.0);
} else if (thrPar.goForward.get() && !thrPar.goRight.get()) {
cout << "UPDATING Y*****\n";
    out_odom.y(thrPar.currentOdo.get().y() + float(y)/100.0);
    out_odom.x(thrPar.currentOdo.get().x());
    out_odom.phi(90*M_PI/180.0);
} else {
cout << "NOT UPDATING ODOMETRY!!!!\n";
out_odom.x(thrPar.currentOdo.get().x());
out_odom.y(thrPar.currentOdo.get().y());
out_odom.phi(thrPar.currentOdo.get().phi());
}*/
//out_odom.phi((float(phi)+90)*M_PI/180.0);
out_odom.phi(90*M_PI/180.0);
cout << format("Odometry:  x=%0.03f,   y=%0.03f,   phi=%0.03f", (float)out_odom.x(),
(float)out_odom.y(), (float)out_odom.phi()) << endl;

sleep(1000);

}

int setupArduino(char * port, int readBytes)
{

/* initialize for hardware readings */
char *portname = port;
int fd;

/* Open the file descriptor in non-blocking mode */
fd = open(portname, O_RDWR | O_NOCTTY);

```

```

/* Set up the control structure */
struct termios toptions;

/* Get currently set options for the tty */
tcgetattr(fd, &toptions);

/* Set custom options */
/* 9600 baud */
cfsetispeed(&toptions, B9600);
cfsetospeed(&toptions, B9600);

/* 8 bits, no parity, no stop bits */
toptions.c_cflag &= ~PARENB;
toptions.c_cflag &= ~CSTOPB;
toptions.c_cflag &= ~CSIZE;
toptions.c_cflag |= CS8;
/* no hardware flow control */
toptions.c_cflag &= ~CRTSCTS;

/* enable receiver, ignore status lines */
toptions.c_cflag |= CREAD | CLOCAL;

/* disable input/output flow control, disable restart chars */
toptions.c_iflag &= ~(IXON | IXOFF | IXANY);

/* disable canonical input, disable echo,
disable visually erase chars,
disable terminal-generated signals */
toptions.c_iflag &= ~(ICANON | ECHO | ECHOE | ISIG);

/* disable output processing */
toptions.c_oflag &= ~OPOST;

/* wait for 1 characters to come in before read returns */
toptions.c_cc[VMIN] = readBytes;

/* no minimum time to wait before read returns */
toptions.c_cc[VTIME] = 0;

/* commit the options */
tcsetattr(fd, TCSANOW, &toptions);

/* Wait for the Arduino to reset */
sleep(1000);
return fd;
}

/*
 * @Description
 * Try to drive our robot smoothly: avoid stop (between steps) if don't need to. Also
 * adjust bearing toward the next step.
 *
 * @param      aPath:  calculated path

```

```

*           aRobot: access to robot odometry
*           thrPar: access and update thread parameter list
* @return   none
*/
static void smoothDrive(CActivMediaRobotBase & aRobot, deque<poses::TPoint2D> aPath,
TThreadRobotParam & thrPar)
{
    CPose2D initOdo;
    int odo_fd = thrPar.odo_fd.get();
    getOdometry(initOdo, odo_fd, thrPar);
    ( initOdo, thrPar.odometryOffset.get() );

    CPose2D currentOdo, previousOdo;
    getOdometry(currentOdo, odo_fd, thrPar);
    fixOdometry( currentOdo, thrPar.odometryOffset.get() );
    thrPar.currentOdo.set(currentOdo);

    getOdometry(previousOdo, odo_fd, thrPar);
    fixOdometry( previousOdo, thrPar.odometryOffset.get() );

    /* for each individual step on path */
    for (deque<poses::TPoint2D>::const_iterator it=aPath.begin(); it!=aPath.end() &&
(thrPar.quit.get() == false); ++it)
    {
        double phi;
        double tempAngle;
        CPose2D tempOdo;

        cout << "current pose: " << currentOdo << endl;
        cout << "[next nearby target?] x: " << it->x << " y: " << it->y << " " << endl;

        //If getting LRF don't move to the next location
        while(thrPar.gettingLRF.get())
        {
            sleep(100);
        }

        thrPar.isMoving.set(true);
        /* check for undefined case atan(0,0), for sure */
        if( ( abs(it->y - currentOdo.y()) < SMALL_NUMBER ) && ( abs(it->x - currentOdo.x())
< SMALL_NUMBER ) )
        {
            thrPar.currentOdo.set(currentOdo);
            cout << "continue....." << endl;
            continue;
        }
        getOdometry( currentOdo,odo_fd, thrPar);//, v, w, left_ticks, right_ticks );
        //cout << " Odometry: " << currentOdo;
        fixOdometry( currentOdo, thrPar.odometryOffset.get() );
        //cout << " Fixed Odometry: " << currentOdo;
        thrPar.currentOdo.set(currentOdo);
        //cout << " Fixed Odometry 2: " << currentOdo << endl;
    }
}

```



```

while( /* lower bound, make sure robot does not pass target */
      ( currentOdo.distanceTo(CPoint2D(it->x,it->y)) > FORWARD_THRESHOLD )
      /* upper bound, make sure robot does not go far off target */
      && ( currentOdo.distanceTo(CPoint2D(it->x,it->y)) < 6 * MIN_STEP )
      /* stop condition from user interface (esc/spacebar key) */
      && (thrPar.quit.get() == false)
      && (thrPar.stop.get() == false)
      )
{
    /* calculate required phi to next nearby target */
    phi = atan2 ( (it->y - currentOdo.y() ) , (it->x - currentOdo.x() ) );
    /* the next dozen lines of code basically determine whether robot
    * should stop and turn or not
    */
    tempOdo = CPose2D(currentOdo);
    tempOdo.normalizePhi();
    tempAngle = abs(phi - tempOdo.phi());
    if ( tempAngle >= M_PI )
    {
        tempAngle = 2*M_PI - tempAngle;
    }
    /* if current phi > SHARP_TURN, stop and turn */
    if ( tempAngle >= SHARP_TURN )
    {
        cout << "phi: " << RAD2DEG(phi);
        cout << "\t tempOdo.phi: " << RAD2DEG(tempOdo.phi()) << endl;
        cout << "turning ... " << endl;
        turn(phi, thrPar);
        cout << "done with turn"<< endl;
        getOdometry(currentOdo,odo_fd, thrPar);
        fixOdometry( currentOdo, thrPar.odometryOffset.get() );
        cout << "Phi after turn: " << currentOdo.phi() << endl;
    }

    /* simple collision avoidance "smooth" drive */

    /* if no obstacle */
    if(
        (thrPar.leftObstacle.get() == false)
        && (thrPar.rightObstacle.get() == false)
        && (thrPar.centerObstacle.get() == false)
        )
    {
        /* driving to next target */
        /* FIXME : adjust ANGULAR_SPEED_DIV for better drive */
        double turn_angle = turnAngle(currentOdo.phi(), phi) /
ANGULAR_SPEED_DIV;
        if (turn_angle < .02)
            turn_angle = 0;
        /* FIXME : segmentation fault happens sometime, probably it has
something
        * to do with the fixOdometry()
        */
        //cout << "after turnangle " << turn_angle << endl;
        setVelocities( 1,0,thrPar);
    }
}

```

```

        //setVelocities(    LINEAR_SPEED,    turn_angle,    thrPar);
//turnAngle(aRobot, phi, thrPar) / 5 );
    }
    else if (thrPar.rightObstacle.get() == true && thrPar.leftObstacle.get() == false)
    {
        /*turn to the left to avoid the wall on the right */
        //setVelocities( LINEAR_SPEED / 2, 0.2, thrPar );
        //FIX THIS LATER OBSTICAL AVOIDANCE
        setVelocities( 0,0,thrPar);
        //sleep(200);
    }
    else if (thrPar.leftObstacle.get() == true && thrPar.rightObstacle.get() == false)
    {
        /* turn to the right to avoid the wall on the left. */
        //setVelocities( LINEAR_SPEED / 2, -0.2, thrPar );
        setVelocities(0,0, thrPar);
        //sleep(200);
    }
    else /* obstacle in front. FIXME: not having any appropriate behaviour for
this */
    {
        cout << "error! I am confused and cannot navigate appropriately at this
time." << endl;

        //thrPar.stop.set(true);
        //setVelocities(0, 0);
    }

    /* update sonar reading to display */
    CObservationRange obs;
    bool thereis;
    aRobot.getSonarsReadings(thereis,obs);
    if (!thereis)
    {
        cout << "Sonars: NO" << endl;
    }
    else
    {
        adjustCObservationRangeSonarPose( obs );
        displaySonars(thrPar, obs);
        //cout << "Sonars: ";
        //for (CObservationRange::const_iterator
i=obs.sensedData.begin();i!=obs.sensedData.end();++i)
        //    cout << i->sensedDistance << " ";
        //cout << endl;
    }

    mrpt::system::sleep(1000);
//    mrpt::system::sleep(POLL_INTERVAL);
    //sleep(1000);
    /* update robot status */
    setVelocities(0,0,thrPar);

    mrpt::system::sleep(1000);

```

```

        getOdometry( currentOdo, odo_fd, thrPar); /* for stop condition */
        fixOdometry( currentOdo, thrPar.odometryOffset.get() );
        thrPar.currentOdo.set(currentOdo);
        //cout << "end while" << endl;
    }/* end while */

    /* FIXME: update robot odometry to mostlikely particle, below is an example, not tested.
    * The issue is in the changeOdometry() function, which does not work properly. Our
current
    * fix for this is to do a mapping using odometryOffset
    */
//      CMonteCarloLocalization2D          *          tempPdf          =
(CMonteCarloLocalization2D*)thrPar.pdf.get().pointer();
//      CPose2D tempPose;
//      tempPdf->getMean( tempPose );
//      CPose2D startOdo;
//      getOdometry (startOdo);
//      startOdo.x(0);
//      startOdo.y(0);
//      //CPose2D startOdo(newX,newY,DEG2RAD(newPhi));
//      CPose2D newOffset = tempPose;
//      newOffset.phi( DEG2RAD(tempPose.phi()) - startOdo.phi() );
//      thrPar.odometryOffset.set(newOffset);
//      aRobot.changeOdometry(startOdo);
//      currentOdo = tempPose;
//
//Stop moving a delay to allow getting LRF values
//setVelocities( 0, 0, thrPar );
thrPar.isMoving.set(false);
sleep(100);
} /* end for */

/* done driving and save outputs */

//setVelocities( 0, 0, thrPar );
thrPar.stop.set(false);
//
}

/*
 * @Description
 * Get current kinect reading
 * @param      lastObs: last observation from kinect
 *              TP:          Access to current param list
 * @return      current kinect reading (CObservation2DRangeScan), null if no new reading
 */
static CObservation2DRangeScan* getKinect2DScan(const TThreadRobotParam & TP,
CObservation3DRangeScanPtr & lastObs)
{
    //CObservation3DRangeScanPtr newObs = TP.new_obs.get();
    //CObservation2DRangeScanPtr obs2D;
/*
    if (newObs && newObs->timestamp!=INVALID_TIMESTAMP &&

```

```

        (!lastObs || newObs->timestamp!=lastObs->timestamp) )
    {
        // It IS a new observation:
        lastObs = newObs;
        //last_obs_imu = thrPar.new_obs_imu.get();
        // Update visualization -----

        // Convert ranges to an equivalent 2D "fake laser" scan:
        if (lastObs->hasRangeImage )
        {
            // FIXME memory leak ?
            CObservation2DRangeScan* obs2D = new CObservation2DRangeScan();
            lastObs->convertTo2DScan(*obs2D, "KINECT_2D_SCAN");
            return obs2D;
        }
    }*/
    /* else return NULL */
    return NULL;
}

/*
 * @Description
 * This used to be part of the turn function, bring it out here for other uses
 * @param    phi:    calculatated phi to nearby target
 *           aRobot: access to robot odometry
 *           thrPar:  access to thread parameter list
 * @return    offset to desired bearing
 */
double turnAngle(CActivMediaRobotBase & aRobot, double phi, TThreadRobotParam thrPar)
{
    CPose2D odoTemp;
    double ret;
    int odo_fd = thrPar.odo_fd.get();
    getOdometry( odoTemp,odo_fd, thrPar);

    fixOdometry( odoTemp, thrPar.odometryOffset.get() );
    odoTemp.normalizePhi();

    /* find the optimized turn angle */
    ret = abs(phi - odoTemp.phi());
    if ( ret >= M_PI )
    {
        ret = 2*M_PI - ret;
    }

    /* find the turn direction: add turnAngle, normalized then compare */

    odoTemp.phi_incr(ret);
    odoTemp.normalizePhi();
    //cout << "after incr" << endl;
    if ( abs(odoTemp.phi() - phi) > SMALL_NUMBER ) /* small diff */
    {
        ret = -ret;
    }
}

```

```

    }
    //cout << "return " << ret << endl;
    return ret;
}

/*
 * @Description
 * overloading method of turnAngle()
 * This used to be part of the turn function, bring it out here for other uses
 *
 * @param    phi:            calculated phi to nearby target
 *           current phi:    current phi of robot
 * @return    offset to desired bearing
 */
double turnAngle(double current_phi, double phi)
{
    CPose2D odoTemp;
    odoTemp.x(0);
    odoTemp.y(0);
    odoTemp.phi(current_phi);
    double ret;

    odoTemp.normalizePhi();

    /* find the optimized turn angle */
    ret = abs(phi - odoTemp.phi());
    if ( ret >= M_PI )
    {
        ret = 2*M_PI - ret;
    }

    /* find the turn direction: add turnAngle, normalized then compare */

    odoTemp.phi_incr(ret);
    odoTemp.normalizePhi();

    if ( abs(odoTemp.phi() - phi) > SMALL_NUMBER ) /* small diff */
    {
        ret = -ret;
    }

    return ret;
}

/*
 * @Description
 * @Description
 * Turn robot direction to target next target
 *
 * @param    phi:    is the desired turn angle, which referenced to the map coordiate,
 *                 not the current "phi" of robot phi must be normalized into range
 *                 [-pi,pi] before using.
 * robot:    access to robot odometry
 */

```

```

*           p:           access and update current odometry to thread parameter list
* @return   none
*/
static void turn( double phi, TThreadRobotParam &p)
{
    CPose2D      odo;
    CPose2D      target;
    double  v,w;
    int64_t  left_ticks, right_ticks;
    double  speed;
    double  turnAngle;
    int odo_fd = p.odo_fd.get();
    while(1)
    {
        getOdometry( odo, odo_fd, p); //, v, w, left_ticks, right_ticks );
        fixOdometry( odo, p.odometryOffset.get() );
        p.currentOdo.set(odo);           // Update current odometry for threads.

        /* force odo.phi() to be in the range [-pi,pi] */
        odo.normalizePhi();
        cout<<"TA "<<phi<<","<<odo.phi()<<endl;
        if ( abs(phi - odo.phi()) < .09 ) //was SMALL_NUMBER
            return;

        /* turn: Only turn toward target, avoid circling if over-turn */
        /* find the optimized turn angle */
        turnAngle = phi; //abs(phi-odo.phi());
        cout<<"M_PI"<<TURN_THRESHOLD<<endl;
        cout<<"TA "<<turnAngle<<endl;
        if ( turnAngle >= M_PI )
        {
            turnAngle = 2*M_PI - turnAngle;
        }
        cout<<"TA "<<turnAngle<<endl;
        /* find the turn direction: add turnAngle, normalized then compare */
        CPose2D odoTemp(odo);
        odoTemp.phi_incr(turnAngle);
        odoTemp.normalizePhi();

        /* check if we reach turn angle */
        if( turnAngle < TURN_THRESHOLD )
            break;

        if ( abs(odoTemp.phi() - phi) < SMALL_NUMBER ) /* small diff */
        {
            cout<<"1"<<endl;
            if( turnAngle < TURN_THRESHOLD * 2 ) /* slow down near desired angle */
                //speed = ANGULAR_SPEED / 2;
                speed = 1;
            else
                //speed = ANGULAR_SPEED;
                speed = 1;
        }
    }
}

```

```

else
{
cout<<"2"<<endl;
    if( turnAngle < TURN_THRESHOLD * 2 )
        //speed = -ANGULAR_SPEED / 2; /* slow down near desired angle */
        speed = -1;
    else
        //speed = -ANGULAR_SPEED;
        speed = -1;
}

//mrpt::system::sleep(500);
/* do turn */
cout<<"Speed"<<speed<<endl;
setVelocities( 0,-1* speed , p );

/* delay between reading */
//mrpt::system::sleep(POLL_INTERVAL);
mrpt::system::sleep(500);
//sleep(250);

setVelocities(0, 0, p); /* stop */
    mrpt::system::sleep(1200);
}
setVelocities(0, 0, p); /* stop */
}

bool returnGettingLRF(TThreadRobotParam &thrPar)
{
    return thrPar.gettingLRF.get();
}

void setVelocities(int linear, int angular, TThreadRobotParam &thrPar)
{
    int velocity_fd = thrPar.vel_fd.get(); //file descriptor for writing to robot velocity
    char velocityCommand[1];
    int n;

    //wait for lrf reading to finish

    while(thrPar.gettingLRF.get());
    /* Flush anything already in the serial buffer */
    tcflush(velocity_fd, TCIFLUSH);
    velocityCommand[0] = 0;
    //STOP
    if(linear == 0 && angular == 0)
    {
        velocityCommand[0] = 'x';
        /* read up to 128 bytes from the fd */
        write(velocity_fd,velocityCommand,1);
        sleep(100);

        write(velocity_fd,velocityCommand,1);
    }
}

```

```

        sleep(100);
        write(velocity_fd,velocityCommand,1);
        sleep(100);
        write(velocity_fd,velocityCommand,1);
        sleep(100);
        write(velocity_fd,velocityCommand,1);
        sleep(100);
    }
    //LEFT
    else if(linear == 0 && angular == -1)
    {
        velocityCommand[0] = 'a';
        /* read up to 128 bytes from the fd */
        write(velocity_fd,velocityCommand,1);
    }
    //RIGHT
    else if(linear == 0 && angular == 1)
    {
        velocityCommand[0] = 'd';
        /* read up to 128 bytes from the fd */
        write(velocity_fd,velocityCommand,1);
    }
    //FORWARD
    else if(linear == 1 && angular == 0)
    {
        velocityCommand[0] = 'w';
        /* read up to 128 bytes from the fd */
        write(velocity_fd,velocityCommand,1);
    }
    //BACK
    else if(linear == -1 && angular == 0)
    {
        velocityCommand[0] = 's';
        /* read up to 128 bytes from the fd */
        write(velocity_fd,velocityCommand,1);
    }
    else
    {
        velocityCommand[0] = 'x';
        /* read up to 128 bytes from the fd */
        write(velocity_fd,velocityCommand,1);
    }
}

/*
 * @Description
 * Thread to display robot and map in a 3D window.
 *
 * @param      p:          thread parameter list
 * @return     none
 */
void thread_display(TThreadRobotParam &p)

```



```

{
    mrpt::opengl::CSetOfObjectsPtr          gl_grid,gl_pdf;

    p.displayClearOldPdf.set(true);
    p.displayClearOldSonar.set(true);
    p.displayClearOldPath.set(true);

    CDisplayWindow3D      win("Example of 3D Scene Visualization - MRPT",640,480);
    COccupancyGridMap2D   the_grid;

    CObservation2DRangeScan last_obs;

    COpenGLScenePtr &theScene = win.get3DSceneAndLock();
    the_grid.loadFromBitmapFile(MAP_FILE,MAP_RESOLUTION
/*,xCentralPixel,yCentralPixel*/);

    {
        /* display map */
        if (!gl_grid) gl_grid = CSetOfObjects::Create();
        gl_grid->clear();
        the_grid.getAs3DObject( gl_grid );
        theScene->insert( gl_grid );
    }

    {
        /* display grid */
        opengl::CGridPlaneXYPtr obj = opengl::CGridPlaneXY::Create(-55,55,-41,41,0,1);
        obj->setColor(0.8,0.8,0.8);
        theScene->insert( obj );
    }

    {
        /* display current pose */
        opengl::CSetOfObjectsPtr obj = opengl::stock_objects::RobotPioneer();
        obj->setPose(p.currentOdo.get());
        obj->setName( "robot" );
        theScene->insert( obj );
    }

    {
        /* display pose by encoder */
        opengl::CSpherePtr obj = opengl::CSphere::Create();
        obj->setColor(1,1,0);
        obj->setRadius(0.1);
        obj->setLocation(0,0,0);
        obj->setName( "encoder");
        theScene->insert( obj );
    }

    {
        /* display pose by sonar */
        opengl::CSpherePtr obj = opengl::CSphere::Create();
        obj->setColor(0,0,1);
        obj->setRadius(0.1);
        obj->setLocation(0,0,0);
        obj->setName( "sonarpos");
        theScene->insert( obj );
    }
}

```

```

{
    /* display target point */
    opengl::CSpherePtr obj = opengl::CSphere::Create();
    obj->setColor(0,1,0);
    obj->setRadius(0.1);
    obj->setLocation(0,0,0);
    obj->setName( "target");
    theScene->insert( obj );
}

{
    /* display sonar readings */
    opengl::CSetOfLinesPtr obj = opengl::CSetOfLines::Create();
    obj->setPose(p.currentOdo.get() );
    obj->setName( "sonars" );
    obj->setColor(1,0,0);
}

{
    /* display pdf mean *FIXME* Arrow not positioned properly. The point is at the base of
the arrow. */
    opengl::CArrowPtr obj = opengl::CArrow::Create(0,0,2, 0,0,0, 0.05, 0.01,0.02, 0,0,0 );
    obj->setPose(p.currentOdo.get());
    obj->setName( "mostlikelyParticle" );
    // uncomment to display particles
    //theScene->insert( obj );
}

// IMPORTANT!!! IF NOT UNLOCKED, THE WINDOW WILL NOT BE UPDATED!
win.unlockAccess3DScene();

win.captureImagesStart();

// Texts:
win.addTextMessage(0.01,0.85, "This is a 2D message", TColorf(1,1,1),"sans",11,
mrpt::opengl::NICE, 0);

win.setCameraElevationDeg( 25.0f );
//win.setCameraProjective(false);

//CDisplayWindowPlots winEKF("Tracking - Extended Kalman Filter",450,400);

//winEKF.setPos(10,10);

//winEKF.axis(-2,20,-10,10); winEKF.axis_equal();

// Create EKF
// -----
CRangeBearing EKF;
EKF.KF_options.method = kfEKFNaive;

EKF.KF_options.verbose = true;
EKF.KF_options.enable_profiler = true;

```

```

bool end = false;
CTicTac timer;
timer.Tic();

float          x=VEHICLE_INITIAL_X,y=VEHICLE_INITIAL_Y,phi=DEG2RAD(-
180),v=VEHICLE_INITIAL_V,w=VEHICLE_INITIAL_W;
//float initial_front_wall;
float dy=0, old_y=0;
float dx=0, old_x=0;
float t=0;
float sonarposition = 0, sonarpositionright = 0;

int counter = 0;
ofstream EKFEstimate;
EKFEstimate.open ("ekf.csv");
EKFEstimate << "y, x, front_sonar, right wall, dy, dx, initial_front_wall_distance,
initial_right_wall_distance, EKF_X, EKF_Y" << endl;

while (!end && win.isOpen() )
{
    const double t = timer.Tac();

    // Move the scene:
    COpenGLScenePtr &theScene = win.get3DSceneAndLock();

    //x+=v*DELTA_TIME*(cos(phi)-sin(phi));
    //y+=v*DELTA_TIME*(sin(phi)+cos(phi));

//x=0;
    //y=p.currentOdo.get().y();
    x = p.currentOdo.get().x();
    y = p.currentOdo.get().y();

/*if (p.goRight.get()==true) {
    x = p.currentOdo.get().x();
    //y = 0;
    dx = x - old_x;
    cout << "*****Going Right*****\n";
}
else if (p.goForward.get()==true) {
    cout << "*****Going Forward*****\n";
    y = p.currentOdo.get().y();
    //x = 0;
    dy = y - old_y;
}*/

dx = x - old_x;
dy = y - old_y;

//

```

```

    phi+=w*DELTA_TIME;

    v+=1.0f*DELTA_TIME*cos(t);
    w=-0.1f*DELTA_TIME*sin(t);

/*
    // Simulate noisy observation:
    float realBearing = atan2( y,x );
    float  obsBearing = realBearing      + BEARING_SENSOR_NOISE_STD *
randomGenerator.drawGaussian1D_normalized();
    //printf("Real/Simulated bearing: %.03f / %.03f deg\n", RAD2DEG(realBearing),
RAD2DEG(obsBearing) );
    win.addTextMessage(0.01, 0.93, format("Real/Simulated bearing: %.03f / %.03f deg\n",
RAD2DEG(realBearing), RAD2DEG(obsBearing) ), TColorf(0,0,1), "mono", 9, mrpt::opengl::NICE,10);

    float realRange = sqrt(square(x)+square(y));
    float  obsRange = max(0.0, realRange  + RANGE_SENSOR_NOISE_STD *
randomGenerator.drawGaussian1D_normalized() );
    //printf("Real/Simulated range: %.03f / %.03f \n", realRange, obsRange );
    win.addTextMessage(0.01, 0.9, format("Real/Simulated range: %.03f / %.03f \n",
realRange, obsRange ), TColorf(0,0,1), "mono", 9, mrpt::opengl::NICE,12);
*/

//EKF.doProcess(DELTA_TIME,obsRange, obsBearing);

float sonar12;
float sonar10;
sonar12 = p.front_wall.get();
sonar10 = p.right_wall.get();

EKF.doProcess(sonar12, sonar10, dy, dx);

/*EKF.getProfiler().enter("PF:complete_step");
PF.executeOn(particles, NULL,&SF); // Process in the PF
EKF.getProfiler().leave("PF:complete_step");*/

// Show EKF state:
CRangeBearing::KFVector EKF_xkk;
CRangeBearing::KFMatrix EKF_pkk;

EKF.getState( EKF_xkk, EKF_pkk ); /**Should be edited?*/

//printf("Real: x=%.03f y=%.03f heading=%.03f v=%.03f w=%.03f\n",x,y,phi,v,w);
//cout << format("KF parameters: x=%.03f, y=%.03f, phi=%.03f, v=%.03f, w=%.03f,
t=%.03f", x,y,phi,v,w, t) << endl;
//cout << format("y=%.03f, front_sonar=%.03f, dy=%.03f, initial front wall
distance=%.03f", y, p.front_wall.get(), dy, initial_front_wall) << endl;
cout << format("y=%.03f, x=%.03f, front_sonar=%.03f, right_sonar=%.03f, dy=%.03f,
dx=%.03f, initial front wall distance=%.03f, initial right wall distance=%.03f", y, x, p.front_wall.get(),
p.right_wall.get(), dy, dx, initial_front_wall, initial_right_wall) << endl;

```

```

        //cout << "EKF: " << EKF_xkk << endl; /** and this I think this should be for x and y */
        cout << "EKF_Y: " << EKF_xkk[0]<<, EKF_X: "<<EKF_xkk[1] << endl; /** and this
I think this should be for x and y. I edited them though!*/
        EKFEstimate << format("%.03f, %.03f, %.03f, %.03f, %.03f, %.03f, %.03f", y, x,
p.front_wall.get(), p.right_wall.get(), dy, dx, initial_front_wall, initial_right_wall) << EKF_xkk[0]<<
EKF_xkk[1] << endl;

        old_y = y; // replace old y with current y
        old_x = x;

        // Display the current gridmap x,y position of the cursor on the screen.
        int mouse_x,mouse_y;
        if (win.getLastMousePosition(mouse_x,mouse_y)) // See also:
getLastMousePositionRay()
        {
            // Get the ray in 3D for the latest mouse (X,Y):
            mrpt::math::TLine3D ray;
            theScene->getViewPort("main")-
>get3DRayForPixelCoord(mouse_x,mouse_y,ray);

            // Create a 3D plane, e.g. Z=0
            const mrpt::math::TPlane
ground_plane(TPoint3D(0,0,0),TPoint3D(1,0,0),TPoint3D(0,1,0));

            // Intersection of the line with the plane:
            mrpt::math::TObject3D inters;
            mrpt::math::intersect(ray,ground_plane, inters);

            // Interpret the intersection as a point, if there is an intersection:
            mrpt::math::TPoint3D inters_pt;
            if (inters.getPoint(inters_pt))
            {
                // Display x,y coordinates and left, center, right collision detect flags.
                string location = format("X,Y Location: %05f, %05f",
(float)inters_pt.x,(float)inters_pt.y );
                if( p.leftObstacle.get() ) location += string ( " LEFT" );
                if( p.centerObstacle.get() ) location += string ( " CENTER" );
                if( p.rightObstacle.get() ) location += string ( " RIGHT" );
                win.addTextMessage(0.01,0.85, location, TColorf(0,0,1),"sans",12,
mrpt::opengl::NICE, 0);
            }
        }

        win.addTextMessage(5,-15, // |X|,|Y|>1 means absolute coordinates, negative means from
the top instead of the bottom.
        format("Time: %s", mrpt::system::dateTimeLocalToString( mrpt::system::now()
).c_str() ),
        TColorf(1,1,1),
        "mono",9, // font name & size
        mrpt::opengl::NICE,

```

```

        20 // An arbitrary ID to always overwrite the same, previous 2D text message
        );

        // Point camera at the current robot position.
        win.setCameraPointingToPoint(p.currentOdo.get().x(),p.currentOdo.get().y(),0);

CPose2D kalman;
kalman.y(EKF_xkk[0]);
kalman.x(EKF_xkk[1]);
kalman.phi(p.currentOdo.get().phi());
        p.currentKF.set(kalman);

        sonarposition = (initial_front_wall - p.front_wall.get());
        sonarpositionright = (initial_right_wall - p.right_wall.get());

        CPose2D sonarpos_;
        sonarpos_.y(sonarposition);
        sonarpos_.x(sonarpositionright);
        sonarpos_.phi(p.currentOdo.get().phi());
        p.currentSonar.set(sonarpos_);

        /* Change pose of robot in display */
        opengl::CRenderizablePtr obj1 = theScene->getByName("robot");
        obj1->setPose( p.currentKF.get() ); //.x() , p.currentOdo.get().y() , 0 );

        /* Change pose of robot based on encoder in display */
        opengl::CRenderizablePtr obj2 = theScene->getByName("encoder");
        obj2->setPose( p.currentOdo.get() ); //.x() , p.currentOdo.get().y() , 0 );
        cout << format("x=%.03f, y=%.03f, phi=%.03f", p.currentOdo.get().x(),
p.currentOdo.get().y(), p.currentOdo.get().phi()) << endl;

        /* Change pose of robot based on sonar in display */
        opengl::CRenderizablePtr obj3 = theScene->getByName("sonarpos");

        obj3->setPose( p.currentSonar.get() ); //.x() , p.currentOdo.get().y() , 0 );

        /* Change location of target marker */
        opengl::CRenderizablePtr obj4 = theScene->getByName("target");
        obj4->setLocation(p.targetOdo.get().x() , p.targetOdo.get().y() , 0 );

        /* Put new path in the display */
        if(p.displayNewPath.get() )
        {
                std::deque<poses::TPoint2D> thePath(p.thePath.get());
                for (std::deque<poses::TPoint2D>::const_iterator
it=thePath.begin();it!=thePath.end();++it)
                {
                        string objName;
                        std::stringstream sstm;
                        sstm << "path" << it->x << it->y;
                        objName = sstm.str();
                        opengl::CSpherePtr obj = opengl::CSphere::Create();
                        obj->setColor(1,0,0);

```

```

        obj->setRadius(0.04);
        obj->setLocation(it->x,it->y,0);
        obj->setName( objName );
        theScene->insert( obj );
    }
    p.displayNewPath.set(false);
}

/* Put new pdf in the display */
if(p.displayNewPdf.get() )
{
    if (!gl_pdf) {
        gl_pdf = CSetOfObjects::Create();
    }
    if ( p.displayClearOldPdf.get() ) gl_pdf->clear();
    CMonteCarloLocalization2D * tempPdf =
(CMonteCarloLocalization2D*)p.pdf.get().pointer();
    tempPdf->getAs3DObject( gl_pdf );
    theScene->insert( gl_pdf );

    opengl::CRenderizablePtr obj4 = theScene->getByName("mostlikelyParticle");
    CPose2D tempPose;
    tempPdf->getMean( tempPose );
    obj4->setPose( tempPose ); //tempPdf->getMostLikelyParticle() );

    p.displayNewPdf.set(false);
}

/* Put new sonars in the display */
if(p.displayNewSonars.get() )
{
    if ( p.displayClearOldSonar.get() )
    {
        opengl::CRenderizablePtr obj3 = theScene->getByName("sonars");
        if (obj3 != NULL) ( theScene->removeObject(obj3) );
    }
    opengl::CSetOfLinesPtr sonar_object = opengl::CSetOfLines::Create();
    sonar_object->setPose(p.currentOdo.get() );
    sonar_object->setName( "sonars" );
    sonar_object->setColor(1,0,0);
    p.displayNewSonars.set( false );
    deque<TSegment3D> sonarLines( p.sonars.get() );
    for (std::deque<TSegment3D>::const_iterator
it=sonarLines.begin();it!=sonarLines.end();++it)
    {
        sonar_object->appendLine( it->point1.x, it->point1.y, .05, it->point2.x,
it->point2.y, .05 );
        //cout << it->point1.x << " " << it->point1.y <<" " << it->point2.x << "
" << it->point2.y << endl;
    }
    //cout << endl;
    theScene->insert( sonar_object );
}

```

```

}

/* Put new kinect ranges in the display */
{
    /* make sure that we have new kinect reading */
    CObservation2DRangeScanPtr obs_Ptr = p.new_obs.get();
    CObservation2DRangeScan* obs_2d = obs_Ptr.pointer();

    /* make sure that we have new kinect reading */
    if (obs_2d != NULL)
    {
        cout << "obs_2d IS NOT NULL!\n";
        obs_2d->truncateByDistanceAndAngle(kinectMinTruncateDistance,5);
        opengl::CRenderizablePtr obj4 = theScene->getByName( "kinect" );
        if (obj4 != NULL) ( theScene->removeObject(obj4) );
        opengl::CPlanarLaserScanPtr kinect_scan =
opengl::CPlanarLaserScan::Create();
        kinect_scan->setScan( *obs_2d );
        //kinect_scan->setPose(p.currentOdo.get() );
        kinect_scan->setPose(p.currentKF.get() );
        kinect_scan->setName( "kinect" );
        kinect_scan->setColor(1,0,0);
        theScene->insert( kinect_scan );
    } else {
        cout << "obs_2d IS NULL!\n";
    }
}

// IMPORTANT!!! IF NOT UNLOCKED, THE WINDOW WILL NOT BE UPDATED!
win.unlockAccess3DScene();

// Update window:
win.forceRepaint();
mrpt::system::sleep(100);

/* Code to save images of the screen for creating movies. */
/* *FIXME* Caused the kinect to work poorly due to the time spent saving files.
// It may help to reduce the frequency of saving the images
// Grab frame:
//mrpt::utils::CImagePtr img = win.getLastWindowImagePtr();
//if (img)
//{
//    static int i=0;
//    const string s = format("GRAB_%06i.png", ++i );
//    img->saveToFile(s);
//    //printf("Saved frame image to: %s \r",s.c_str() ); // "\ r" is to overwrite the same
line over and over again..
//}
*/

/* Get key press events while 3D window is active. */
if (win.keyHit())
{

```



```

mrptKeyModifier kmods;
int key = win.getPushedKey(&kmods);
printf("Key pushed: %c (%i) - modifiers: 0x%04X\n",char(key),key,kmods);

p.pushed_key.set(key);

switch (key)
{
    case MRPTK_ESCAPE:
        end = true;
        p.quit.set(true);
        break;

    case MRPTK_RIGHT:
        win.setCameraAzimuthDeg( win.getCameraAzimuthDeg() + 5
);
        break;

    case MRPTK_LEFT:
        win.setCameraAzimuthDeg( win.getCameraAzimuthDeg() - 5
);
        break;

    case MRPTK_UP:
        win.setCameraElevationDeg( win.getCameraElevationDeg() +
5 );
        break;

    case MRPTK_DOWN:
        win.setCameraElevationDeg( win.getCameraElevationDeg() -
5 );
        break;

    /* Note: changing camera pointing to point is temporary because the
    pointing to point is changed to current robot odometry each loop. */
    case 'w':
    case 'W':
        {
            float x,y,z;
            win.getCameraPointingToPoint( x,y,z);
            win.setCameraPointingToPoint( x , y + 1, z );
            break;
        }

    case 'a':
    case 'A':
        {
            float x,y,z;
            win.getCameraPointingToPoint( x,y,z);
            win.setCameraPointingToPoint( x - 1, y , z );
            break;
        }
}

```

```

        }

        case 'x':
        case 'X':
            {
                float x,y,z;
                win.getCameraPointingToPoint( x,y,z);
                win.setCameraPointingToPoint( x , y - 1, z );
                break;
            }

        case 'd':
        case 'D':
            {
                float x,y,z;
                win.getCameraPointingToPoint( x,y,z);
                win.setCameraPointingToPoint( x + 1, y , z );
                break;
            }

        /* clear / do not clear previous pdf before adding new pdf to display */
        case 'p':
            p.displayClearOldPdf.set(true);
            break;

        case 'P':
            p.displayClearOldPdf.set(false);
            break;

        /* clear / do not clear previous sonar readings before adding new pdf to
display */
        case 's':
            p.displayClearOldSonar.set(true);
            break;

        case 'S':
            p.displayClearOldSonar.set(false);
            break;

        /* stops current path finding navigation */
        case ' ':
            p.stop.set(true);
            break;
            // end case
        }

    }

}

EKFEstimate.close();

```

```

}

// -----
//                               TestPathPlanning
// -----
static int PathPlanning(std::deque<poses::TPoint2D> &thePath, CPoint2D origin, CPoint2D target)
{
    //string myGridMap( string("FAB-LL-Central-200px.png") );
    float resolution = MAP_RESOLUTION; // size of the grid in meters
    float xCentralPixel = X_CENTRAL_PIXEL; // x central pixel
    float yCentralPixel = Y_CENTRAL_PIXEL; // y central pixel // Load
the gridmap:

    COccupancyGridMap2D gridmap;

    if (!mrpt::system::fileExists(MAP_FILE))
        THROW_EXCEPTION_CUSTOM_MSG1("Map file '%s' not
found",MAP_FILE.c_str());

    printf("Loading gridmap...");
    //CFileGZInputStream(myGridMap) >> gridmap;
    gridmap.loadFromBitmapFile(MAP_FILE,resolution ,xCentralPixel,yCentralPixel);
    printf("Done! %f x %f m\n", gridmap.getXMax()-gridmap.getXMin(), gridmap.getYMax()-
gridmap.getYMin());

    // Find path:
    CPathPlanningCircularRobot pathPlanning;
    pathPlanning.robotRadius = ROBOT_RADIUS;
    pathPlanning.minStepInReturnedPath = MIN_STEP;

    //std::deque<poses::TPoint2D> thePath;
    bool notFound;
    CTicTactictac;

    cout << "Origin: " << origin << endl;
    cout << "Target: " << target << endl;

    cout << "Searching path..."; cout.flush();
    tictac.Tic();
    pathPlanning.computePath( gridmap, origin, target, thePath, notFound, 1000.0f /* Max. distance */
);

    double t = tictac.Tac();
    cout << "Done in " << t*1000 << " ms" << endl;

    printf("Path found: %s\n", notFound ? "NO":"YES");
    printf("Path has %u steps\n", (unsigned)thePath.size());

    // Save result:

```

```

    QImage          img;
    gridmap.getAsImage(img,false, true); // Force a RGB image

    const std::string dest = "path_planning1.png";
    cout << "Saving output to: " << dest << endl;
    img.saveToFile(dest);
    printf("Done\n");

    // Draw the path:
    // -----
    int R = round(pathPlanning.robotRadius / gridmap.getResolution() );

    for (std::deque<poses::TPoint2D>::const_iterator it=thePath.begin();it!=thePath.end();++it)
    {
        img.drawCircle(    gridmap.x2idx(it->x),gridmap.getSizeY()-1-gridmap.y2idx(it->y),R,
TColor(0,0,255) );
//        cout << "x: " << it->x << " y: " << it->y << " " << endl;
    }
    img.cross(gridmap.x2idx(origin.x()),gridmap.getSizeY()-1-
gridmap.y2idx(origin.y()),TColor(0x20,0x20,0x20),'+',10);
    img.cross(gridmap.x2idx(target.x()),gridmap.getSizeY()-1-
gridmap.y2idx(target.y()),TColor(0x50,0x50,0x50),'x',10);

    const std::string dest2 = "path_planning2.png";
    cout << "Saving output to: " << dest2 << endl;
    img.saveToFile(dest2);
    printf("Done\n");

    return notFound ? -1:1;
#if MRPT_HAS_WXWIDGETS
//    mrpt::gui::CDisplayWindow    win("Computed path");
//    win.showImage(img);

//    win.waitForKey();
#endif

}

/*
 * @Description
 * Set up the locations and headings of the sonars for simulating sonar in the map.
 *
 * @param    obs:    the sonar readings to be set.
 * @return    none
 */
void createCObservationRange( CObservationRange &obs )
{
    obs.minSensorDistance = 0;
    obs.maxSensorDistance = 7;

    int          i,N = 16;

```

```

obs.sensorLabel = "BASE_SONARS";
obs.sensorConeApperture = DEG2RAD( 30 );
obs.timestamp = system::now();

obs.sensedData.clear();
float yawVals [] = { 90.00,50.00, 30.00, 10.00, -10.00, -30.00, -50.00, -90.00, -90.00,
-130.00, -150.00, -170.00, 170.00, 150.00, 130.00, 90.00 };
float xVals [] = { 0.069, 0.114, 0.148, 0.166, 0.166, 0.148, 0.114, 0.069, -0.02,
-0.024, -0.058, -0.077, -0.077, -0.058, -0.024, -0.02 };
float yVals [] = { 0.136, 0.119, 0.078, 0.027, -0.027, -0.078, -0.119, -0.136, -0.136,
-0.119, -0.078, -0.027, 0.027, 0.078, 0.119, 0.136 };
for (i=0;i<N;i++)
{

    obs.sensedData.push_back( CObservationRange::TMeasurement() );
    CObservationRange::TMeasurement & newObs = obs.sensedData.back();

    newObs.sensorID = i;
    newObs.sensorPose.x = xVals[i];
    newObs.sensorPose.y = yVals[i];
    newObs.sensorPose.z = 0;
    newObs.sensorPose.yaw = DEG2RAD( yawVals[i] );
    newObs.sensorPose.pitch = 0;
    newObs.sensorPose.roll = 0;

    newObs.sensedDistance = 2;

}

}

/*
 * @Description
 * Create a deque of line segments from a sonar reading. To be used by the display thread.
 *
 * @param p: the thread parameter struct.
 *          sonars: the sonar reading to be converted.
 * @return none
 */
void displaySonars(TThreadRobotParam &p, CObservationRange &sonars)
{
    deque<TSegment3D> sonarLines;
    for (CObservationRange::const_iterator
i=sonars.sensedData.begin();i!=sonars.sensedData.end();++i)
    {
        //TSegment3D line;
        TPose3D sPos;
        sPos = i->sensorPose; // set sPos as sensor pose.
        //sPos.x = sPos.x + p.currentOdo.get().x(); // add robot pose to sensor pose.
        //sPos.y = sPos.y + p.currentOdo.get().y(); // add robot pose to sensor pose.
        TPoint3D p1,p2; // starting and
ending points of line segment
        p1.x = sPos.x;

```

```

        p1.y = sPos.y;
        p1.z = 0; // set first point to
sensor pose
        p2.x = sPos.x + cos( sPos.yaw ) * i->sensedDistance;
        p2.y = sPos.y + sin( sPos.yaw ) * i->sensedDistance;
        p2.z = 0;
        TSegment3D line( p1,p2 );
        sonarLines.push_back( line );
        //cout << "p1: " << p1 << " p2: " << p2 << endl;
    }

    //cout << "displaySonars: ";
    //for (std::deque<TSegment3D>::const_iterator
it=sonarLines.begin();it!=sonarLines.end();++it)
        {
        // cout << it->point1 << " ";
        // }
        //cout << endl;
    p.sonars.set( sonarLines );
    p.displayNewSonars.set( true );

}

/*
 * @Description
 * This function will adjust the poses for the sonars of the
 * PeopleBot to the correct locations and angles.
 * The poses returned by the PEOPLE-Bot are wrong.
 * I believe the error is that the Peoplebot thinks the back
 * sonars are the upper ring, so they are facing forwards.
 *
 * @param obs: the sonar readings to be fixed.
 * @return none
 */
void adjustCObservationRangeSonarPose( CObservationRange &obs )
{
    int sensor = 0;
    TPose3D sensor_pose;

    float yawVals [] = { 90.00,50.00, 30.00, 10.00, -10.00, -30.00, -50.00, -90.00, -90.00,
-130.00, -150.00, -170.00, 170.00, 150.00, 130.00, 90.00 };
    float xVals [] = { 0.069, 0.114, 0.148, 0.166, 0.166, 0.148, 0.114, 0.069, -0.02,
-0.024, -0.058, -0.077, -0.077, -0.058, -0.024, -0.02 };
    float yVals [] = { 0.136, 0.119, 0.078, 0.027, -0.027, -0.078, -0.119, -0.136, -0.136,
-0.119, -0.078, -0.027, 0.027, 0.078, 0.119, 0.136 };
    float sensed_distance [] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };

    for (CObservationRange::const_iterator i=obs.sensedData.begin();i!=obs.sensedData.end();++i)
    {
        sensed_distance[sensor] = i->sensedDistance;
        sensor ++;
    }

    sensor = 0;

```

```

int i,N = 16;
obs.sensedData.clear();
for (i=0;i<N;i++)
{

    obs.sensedData.push_back( CObservationRange::TMeasurement() );
    CObservationRange::TMeasurement & newObs = obs.sensedData.back();

    newObs.sensorID = i;
    newObs.sensorPose.x = xVals[i];
    newObs.sensorPose.y = yVals[i];
    newObs.sensorPose.z = 0;
    newObs.sensorPose.yaw = DEG2RAD( yawVals[i] );
    newObs.sensorPose.pitch = 0;
    newObs.sensorPose.roll = 0;

    newObs.sensedDistance = sensed_distance[sensor];
    sensor++;
}

}

/*
 * @Description
 * This function computes the likelihood of the observation given the pose and map.
 * The mrpt library likelihood function only works with laser scans, sonar scans
 * are not supported.
 *
 * @param      map:    the map for checking the sonar readings.
 *              pose:   pose that current sonar readings will be checked at.
 *              obs:    the sonar readings to be checked.
 * @return      none
 */
double CObservationRangeLikelihood(COccupancyGridMap2D & map, CPose2D & pose,
CObservationRange & obs)
{
    CObservationRange simObs;
    float sensed_distance [] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };
    float simulated_distance [] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };
};

int sensor = 0;
double return_value = 1.0;
int valid = 0;

for (CObservationRange::const_iterator i=obs.sensedData.begin();i!=obs.sensedData.end();++i)
{
    sensed_distance[sensor] = i->sensedDistance;
    sensor ++;
}

map.sonarSimulator(
    simObs,          // CObservationRange &   inout_observation,
    pose,           // current pose

```

```

        0.5f,          // float threshold = 0.5f
        0,           // float rangeNoiseStd = 0,
        DEG2RAD(0) // float angleNoiseStd = DEG2RAD(0)
    );

    sensor = 0;
    for (int i=simObs.sensedData.begin();i!=simObs.sensedData.end();i++) (CObservationRange::const_iterator
    {
        simulated_distance[sensor] = i->sensedDistance;
        sensor ++;
    }

    for (int i=0;i< sensor;i++)
    {
        if ( sensed_distance[i] > .05 && simulated_distance[i] > .05 )
        {
            if ( sensed_distance[i] > simulated_distance[i] )
            {
                return_value = return_value * ( sensed_distance[i] /
simulated_distance[i] );
            }
            else
            {
                return_value = return_value * ( simulated_distance[i] /
sensed_distance[i] );
            }
            valid++;
        }
    }

    if (valid > (sensor * .75) ) return return_value * valid / sensor;
    else return 0.0;
}

/*
 * @Description
 * *FIXME* Nonfunctional - the particle weights are not correctly set by this function.
 * This function uses the CObservationRangeLikelihood function to calculate likelihood
 * of each pose in the pdf with the current sonar readings. Setting weights does not
 * currently work.
 *
 * @param map: the map for checking the sonar readings.
 * pdf: the list of possible robot poses to be compared to the current sonar
readings.
 * obs: the sonar readings to be checked.
 * @return none
 */
void computePdfLikelihoodValues(COccupancyGridMap2D & map, CMonteCarloLocalization2D & pdf,
CObservationRange & obs)
{
    int numberOfParticles = pdf.particlesCount();

```





```

//left

int    left_count = 0,
       center_count = 0,
       right_count = 0;
float left_sum = 0,
       center_sum = 0,
       right_sum = 0;
int left_offset = 50,
    center_offset = 340,
    right_offset = 680;
int stop = 50;

for (int i = 0; i < stop; i+=10)
{
    if(obs_2d->scan[i+left_offset] > KinectMinTruncateDistance)
    {
        left_count += 1;
        left_sum += obs_2d->scan[i+left_offset];
    }
    if(obs_2d->scan[i+center_offset] > KinectMinTruncateDistance)
    {
        center_count += 1;
        center_sum += obs_2d->scan[i+center_offset];
    }
    if(obs_2d->scan[i+right_offset] > KinectMinTruncateDistance)
    {
        right_count += 1;
        right_sum += obs_2d->scan[i+right_offset];
    }
}

if(left_count > 0)
{
    //cout << "Left: " << left_sum / left_count << " ";
    p.leftObstacle.set( ( left_sum / left_count ) > 1 ? false : true );
}
if(center_count > 0)
{
    //cout << "Center: " << center_sum / center_count << " ";
    p.centerObstacle.set( ( center_sum / center_count ) > 1 ? false : true );
} else p.centerObstacle.set( false );
if(right_count > 0)
{
    //cout << "Right: " << right_sum / right_count << endl;
    p.rightObstacle.set( ( right_sum / right_count ) > 1 ? false : true );
}
//else cout << endl;
}

sleep(300);
}

```

```

}

/*
 * @Description
 * Adjusts the given pose by rotating by phi given in offset and then adds the offset
 * x and y values.
 *
 * @param      pose:      the pose to be adjusted.
 *              offset:    the pose containing the x,y, phi offset.
 * @return      none
 */
void fixOdometry(CPose2D & pose, CPose2D offset)
{
    double x = pose.x();
    double y = pose.y();
    double phi = offset.phi();
    //cout << "fixOdometry x 1: " << x << " y: " << y << " z: " << phi << endl;

    double xprime = x * cos (phi) - y * sin (phi);
    double yprime = y * cos (phi) + x * sin (phi);

    pose.x( xprime + offset.x() );
    pose.y( yprime + offset.y() );
    double newPhi = pose.phi() + phi;
    pose.phi( newPhi );

    //cout << "fixOdometry x 2: " << pose.x() << " y: " << pose.y() << " z: " << pose.phi() << endl;
    pose.normalizePhi ();
}

```

```

int clientCommunication()
{
    int sock;
    struct sockaddr_in server;
    char message[1000] , server_reply[2000];

    //Create socket
    sock = socket(AF_INET , SOCK_STREAM , 0);
    if (sock == -1)
    {
        printf("Could not create socket");
    }
    puts("Socket created");

    server.sin_addr.s_addr = inet_addr(LOCAL_HOST);
    server.sin_family = AF_INET;
    //server.sin_port = htons( 8888 );
    server.sin_port = htons( 80 );
}

```

```

//Connect to remote server
if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
{
    perror("connect failed. Error");
    return 1;
}

puts("Connected\n");

//keep communicating with server
while(1)
{
    printf("Enter message : ");
    scanf("%s" , message);

    //Send some data
    //if( send(sock , message , strlen(message) , 0) < 0)
    //{
    //    puts("Send failed");
    //    return 1;
    //}

    //Receive a reply from the server
    if( recv(sock , server_reply , 2000 , 0) < 0)
    {
        puts("recv failed");
        break;
    }

    // server_reply (char [2000]) is the response (a string) from the server
    /*
        The message format would be like this:
            <target>:<command>:<input argument1>:<input argument2>

        The navigation should respond to messages like these from the server:
        - Navigate to a location: 'nav:goto:fab70', 'nav:goto:eb84'
        - Short/small movements: 'nav:move:forward:1', 'nav:move:left:2' (maybe 1 = 30
degrees, 2 = 60 degrees, etc.)

        You might need to check the stringmatching.c file from the Project page > Files
to see how to use regular expression to parse the message.
    */

    puts("Server reply :");
    puts(server_reply);

    parseServerReply(server_reply);
    // Clear the server_reply array
    memset(server_reply, 0, sizeof server_reply);

    printf("x");
}

```

```

        close(sock);
        return 0;
    }

int parseServerReply(char * server_reply)
{
    //Function to parse the server reply
}

CRangeBearing::CRangeBearing()
{
    //KF_options.method = kfEKFNaive;
    KF_options.method = kfEKFAlaDavison;

    // INIT KF STATE
    /*m_xkk.resize(4,0);    // State: (x,y,heading,v,w)
    m_xkk[0]= VEHICLE_INITIAL_X;
    m_xkk[1]= VEHICLE_INITIAL_Y;
    m_xkk[2]=-VEHICLE_INITIAL_V;
    m_xkk[3]=0;
    */
    m_xkk.resize(2,0);
    m_xkk[0]= VEHICLE_INITIAL_Y; // for y
    m_xkk[1]= VEHICLE_INITIAL_X; // for x

    // Initial cov: Large uncertainty
    /*m_pkk.setSize(4,4);
    m_pkk.unit();
    m_pkk(0,0)=
    m_pkk(1,1)= square( 5.0f );
    m_pkk(2,2)=
    m_pkk(3,3)= square( 1.0f );*/
    m_pkk.setSize(2,2);
    m_pkk.unit();
    m_pkk(0,0)=square( 5.0f );
    m_pkk(1,1)=square( 5.0f );
}

CRangeBearing::~CRangeBearing()
{
}

/*void    CRangeBearing::doProcess(    double    DeltaTime,    double    observationRange,    double
observationBearing )
{
    m_deltaTime = (float)DeltaTime;
    m_obsBearing = (float)observationBearing;
}

```

```

        m_obsRange = (float) observationRange;

        runOneKalmanIteration();
    }*/

void CRangeBearing::doProcess( double sonar12, double sonar10, double dy , double dx) {

    m_sonar12 = (float)sonar12;
    m_sonar10 = (float)sonar10;
    m_dy = (float)dy;
    m_dx = (float)dx;

    runOneKalmanIteration();
}

/** Must return the action vector u.
 * \param out_u The action vector which will be passed to OnTransitionModel
 */
void CRangeBearing::OnGetAction( KFFArray_ACT &u ) const
{
    u[0] = 1;
}

/** Implements the transition model  $\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k)$ 
 * \param in_u The vector returned by OnGetAction.
 * \param inout_x At input has  $\hat{x}_{k-1|k-1}$ , at output must have  $\hat{x}_{k|k-1}$ .
 * \param out_skip Set this to true if for some reason you want to skip the prediction step (to do not modify
either the vector or the covariance). Default:false
 */
void CRangeBearing::OnTransitionModel(
    const KFFArray_ACT &in_u,
    KFFArray_VEH &inout_x,
    bool &out_skipPrediction
) const
{
    // in_u[0] : Delta time
    // inout_x: [0]:x [1]:y [2]:vx [3]:vy
    //inout_x[0] += in_u[0] * inout_x[2];
    //inout_x[1] += in_u[0] * inout_x[3];

    inout_x[0] += in_u[0]*(m_dy); // changed by omar, + is removed look at the paper! FOR Y
    inout_x[1] += in_u[0]*(m_dx); // For X
    //This should be:   inout_x[0] = y that comes from the encoder. as the state equals y from encoder.
}

/** Implements the transition Jacobian  $\frac{\partial f}{\partial x}$ 
 * \param out_F Must return the Jacobian.
 * The returned matrix must be  $N \times N$  with N being either the size of the whole state vector or
get_vehicle_size().
 */
void CRangeBearing::OnTransitionJacobian(KFFMatrix_VxV &F) const

```

```

{
    F.unit();

    //F(0,2) = m_deltaTime;
    //F(1,3) = m_deltaTime;
    F(0,0) = 1; //changed by omar // THIS IS the A matrix
    F(1,1) = 1;
}

/** Implements the transition noise covariance \f$ Q_k \f$
 * \param out_Q Must return the covariance matrix.
 * The returned matrix must be of the same size than the jacobian from OnTransitionJacobian
 */
void CRangeBearing::OnTransitionNoise(KFMatrix_VxV &Q) const
{
    /*Q(0,0) =
    Q(1,1) = square( TRANSITION_MODEL_STD_XY );
    Q(2,2) =
    Q(3,3) = square( TRANSITION_MODEL_STD_VXY );
    */
    Q(0,0) = 14; //changed by omar
    Q(1,1) = 14;
}

/** Return the observation NOISE covariance matrix, that is, the model of the Gaussian additive noise of the
sensor.
 * \param out_R The noise covariance matrix. It might be non diagonal, but it'll usually be.
 * \note Upon call, it can be assumed that the previous contents of out_R are all zeros.
 */
void CRangeBearing::OnGetObservationNoise(KFMatrix_OxO &R) const
{
    /*R(0,0) = square( BEARING_SENSOR_NOISE_STD );
    R(1,1) = square( RANGE_SENSOR_NOISE_STD );
    */
    R(0,0) = 12; // changed by omar
    R(1,1) = 12;
}

void CRangeBearing::OnGetObservationsAndDataAssociation(
    vector_KFArray_OBS &out_z,
    mrpt::vector_int &out_data_association,
    const vector_KFArray_OBS &in_all_predictions,
    const KFMatrix &in_S,
    const vector_size_t &in_lm_indices_in_S,
    const KFMatrix_OxO &in_R
)
{
    //out_z.resize(1);
    //out_z[0][0] = m_obsBearing;
    //out_z[0][1] = m_obsRange;
    out_z.resize(1);
    out_z[0][0] = m_sonar12;
    out_z[0][1] = m_sonar10;
}

```

```

        out_data_association.clear(); // Not used
    }

/** Implements the observation prediction  $h_i(x)$ .
 * \param idx_landmark_to_predict The indices of the landmarks in the map whose predictions are expected
 as output. For non SLAM-like problems, this input value is undefined and the application should just generate
 one observation for the given problem.
 * \param out_predictions The predicted observations.
 */
void CRangeBearing::OnObservationModel(
    const vector_size_t &idx_landmarks_to_predict,
    vector_KFArray_OBS &out_predictions
) const
{
    // predicted bearing:
    /*kftype x = m_xkk[0];
    kftype y = m_xkk[1];

    kftype h_bear = atan2(y,x);
    kftype h_range = sqrt(square(x)+square(y));

    // idx_landmarks_to_predict is ignored in NON-SLAM problems
    out_predictions.resize(1);
    out_predictions[0][0] = h_bear;
    out_predictions[0][1] = h_range;
    */
    kftype y = m_xkk[0]; //OMAR, this is the state, and in our case the state is the encoder measurement.
    kftype x = m_xkk[1];

    kftype h_distance = initial_front_wall - y;
    kftype h_distancex = initial_right_wall - x;

    out_predictions.resize(1);
    out_predictions[0][0] = h_distance;
    out_predictions[0][1] = h_distancex;
}

/** Implements the observation Jacobians  $\frac{\partial h_i}{\partial x}$  and (when applicable)  $\frac{\partial h_i}{\partial y_i}$ .
 * \param idx_landmark_to_predict The index of the landmark in the map whose prediction is expected as
 output. For non SLAM-like problems, this will be zero and the expected output is for the whole state vector.
 * \param Hx The output Jacobian  $\frac{\partial h_i}{\partial x}$ .
 * \param Hy The output Jacobian  $\frac{\partial h_i}{\partial y_i}$ .
 */
void CRangeBearing::OnObservationJacobians(
    const size_t &idx_landmark_to_predict,
    KFMatrix_OxV &Hx,
    KFMatrix_OxF &Hy
) const
{
    // predicted bearing:

```



```

    /*kftype x = m_xkk[0];
    kftype y = m_xkk[1];

    Hx.zeros();
    Hx(0,0) = -y/(square(x)+square(y));
    Hx(0,1) = 1/(x*(1+square(y/x)));

    Hx(1,0) = x/sqrt(square(x)+square(y));
    Hx(1,1) = y/sqrt(square(x)+square(y));*/

    // Hy: Not used

    Hx.zeros();
    Hx(0,0) = -1; //changed by omar
    Hx(1,1) = -1;
}

/** Computes A=A-B, which may need to be re-implemented depending on the topology of the individual
    scalar components (eg, angles).
    */
void CRangeBearing::OnSubtractObservationVectors(KFArray_OBS &A, const KFArray_OBS &B) const
{
    A -= B;
    math::wrapToPiInPlace(A[0]); // The angular component
}

/*****
*****/
/*
                MAIN THREAD
*/
/*****
*****/
/*
* @Description
* After starting the threads for kinect, pdf, display and collision avoidance,
* loop while accepting manual commands.
* command menu:
*   - a/d : +/- angular speed
*   - space : stop current motion
*   - o : Query odometry (display odometry is only updated here)
*   - n : Query sonars
*   - b : Query battery level
*   - p : Query bumpers
*   - P : Follow Path (manual control is not available during navigation)
*   - e : Enter new current pose
*   - t : Enter new target for path Planning
*   - x : Quit
*
* @param pose: the pose to be adjusted.
* offset: the pose containing the x,y, phi offset.
* @return none
*/
int main(int argc, char **argv)

```

```

{
    try
    {
        /* read parameters from config file */
        ASSERT_(fileExists(CONFIG_FILE_NAME))
        CConfigFile guidebotConfFile(CONFIG_FILE_NAME);

        TTY_PORT =
guidebotConfFile.read_string("NavigationParams", "TTY_PORT", "/dev/ttyACM0", true);
        COM_PORT =
guidebotConfFile.read_string("NavigationParams", "COM_PORT", "COM4", true);
        BAUD_RATE =
guidebotConfFile.read_int("NavigationParams", "BAUD_RATE", 9600, true);
        ROBOT_RADIUS =
guidebotConfFile.read_float("NavigationParams", "ROBOT_RADIUS", 0.30f, true);
        MIN_STEP = guidebotConfFile.read_float("NavigationParams", "MIN_STEP", 0.40f,
true);
        TURN_THRESHOLD =
guidebotConfFile.read_double("NavigationParams", "TURN_THRESHOLD", M_PI/90, true);
        SHARP_TURN =
guidebotConfFile.read_double("NavigationParams", "SHARP_TURN", M_PI/5, true);
        FORWARD_THRESHOLD =
guidebotConfFile.read_double("NavigationParams", "FORWARD_THRESHOLD", 0.09, true);
        POLL_INTERVAL =
guidebotConfFile.read_int("NavigationParams", "POLL_INTERVAL", 100, true);
        ANGULAR_SPEED =
guidebotConfFile.read_double("NavigationParams", "ANGULAR_SPEED", 0.3, true);
        LINEAR_SPEED =
guidebotConfFile.read_double("NavigationParams", "LINEAR_SPEED", 0.2, true);
        SMALL_NUMBER =
guidebotConfFile.read_double("NavigationParams", "SMALL_NUMBER", 0.001, true);
        ANGULAR_SPEED_DIV =
guidebotConfFile.read_int("NavigationParams", "ANGULAR_SPEED_DIV", 5, true);
        LINEAR_SPEED_DIV =
guidebotConfFile.read_int("NavigationParams", "LINEAR_SPEED_DIV", 2, true);

        MAP_FILE =
guidebotConfFile.read_string("NavigationParams", "MAP_FILE", "FAB-LL-Central-
200px.png", true);
        MAP_RESOLUTION =
guidebotConfFile.read_float("NavigationParams", "MAP_RESOLUTION", 0.048768f, true);
        X_CENTRAL_PIXEL =
guidebotConfFile.read_int("NavigationParams", "X_CENTRAL_PIXEL", -30, true);
        Y_CENTRAL_PIXEL =
guidebotConfFile.read_int("NavigationParams", "Y_CENTRAL_PIXEL", 6, true);

        kinectMinTruncateDistance =
guidebotConfFile.read_float("NavigationParams", "kinectMinTruncateDistance", .5f , true);

        TURN_THRESHOLD = DEG2RAD(TURN_THRESHOLD);
        SHARP_TURN = DEG2RAD(SHARP_TURN);

        /* our robot base object REMOVED FOR MCECS*/

```

```

        CActivMediaRobotBase robot;
/*
#ifdef MRPT_OS_WINDOWS
        string port= COM_PORT;
#else
        string port= TTY_PORT;
#endif

        int port_baud = BAUD_RATE;

        cout << "Setting serial port to: " << port << " @ " << port_baud << endl;
        robot.setSerialPortConfig( port, port_baud );

        // -----
        // Init comms:
        // -----
        robot.enableSonars();
        robot.initialize();
*/

        double cur_v = 0;
        double cur_w = 0;

        CActivMediaRobotBase::TRobotDescription robInfo;
        robot.getRobotInformation(robInfo);

        CPoint2D target( -29, 8); // target for path planning.

        //CPoint2D origin( -29, 10 ); // origin for path planning.

//        cout << "Robot # front bumpers : " << robInfo.nFrontBumpers << endl;
//        cout << "Robot # rear bumpers : " << robInfo.nRearBumpers << endl;
//        cout << "Robot # sonars : " << robInfo.nSonars << endl;

/* -----
 * Launch threads
 * -----
 */
        TThreadRobotParam thrPar;
        mrpt::system::TThreadHandle pdfHandle;
        mrpt::system::TThreadHandle displayHandle;
        mrpt::system::TThreadHandle thHandle;
        //mrpt::system::TThreadHandle wallDetectHandle;

        //setup arduino for odo
        cout<<"Initializing odometry arduino"<<endl;
        int odo_fd = setupArduino(ODO_PORT_NAME,ODO_READ_MIN);
        thrPar.odo_fd.set(odo_fd);
        cout<<"odo_fd: "<<odo_fd<<endl;
        //if lrf and odometry are on the same arduino port
        if(strcmp(ODO_PORT_NAME,LRF_PORT_NAME) == 0)

```

```

    {
        cout<<"Initializing lrf arduino on the same port as odometry"<<endl;
        thrPar.lrf_fd.set(odo_fd);
    }
else
    {
        cout<<"Initializing lrf arduino on unique port"<<endl;
        int lrf_fd = setupArduino(LRF_PORT_NAME,LRF_READ_MIN);
        thrPar.lrf_fd.set(lrf_fd);
    }

//if motor is also on the same port
if(strcmp(ODO_PORT_NAME,MOTOR_PORT_NAME) == 0)
    {
        cout<<"Initializing motor arduino on the same port as odometry"<<endl;
        thrPar.vel_fd.set(odo_fd);
    }
else
    {
        cout<<"Initializing vel arduino on unique port"<<endl;
        int vel_fd = setupArduino(MOTOR_PORT_NAME,1);
        thrPar.vel_fd.set(vel_fd);
    }

//Robot is not in motion, or getting LRF value
thrPar.isMoving.set(false);
thrPar.isTurning.set(false);
thrPar.gettingLRF.set(false);

//pdfHandle = mrpt::system::createThreadRef(thread_update_pdf ,thrPar);
displayHandle = mrpt::system::createThreadRef(thread_display ,thrPar);
thHandle = mrpt::system::createThreadRef(thread_LRF ,thrPar);
//wallDetectHandle = mrpt::system::createThreadRef(thread_wall_detect, thrPar);
/* Wait until data stream starts so we can say for sure the sensor has been initialized OK:
*/
cout << "Waiting for sensor initialization...\n";
/* May need to do similar for LRF
do
{
    CObservation3DRangeScanPtr possiblyNewObs = thrPar.new_obs.get();
    if (possiblyNewObs && possiblyNewObs->timestamp!=INVALID_TIMESTAMP)
        break;
    else
        mrpt::system::sleep(10);
} while (!thrPar.quit);
*/
/* Check error condition: */
if (thrPar.quit.get())
    return 0;

bool show_menu = true;

```

```

int counter = 0;

while (1)
{
    if (show_menu)
    {
        show_menu=false;
        cout << "Press the key for your option:" << endl << endl;
        cout << " w/s  : +/- forward or back" << endl;
        cout << " a/d  : +/- left or right" << endl;
        cout << " space : stop" << endl;
        cout << " o   : Query odometry" << endl;
        cout << " n   : Query sonars" << endl;
        cout << " b   : Query battery level" << endl;
        cout << " p   : Query bumpers" << endl;
        cout << " P           : Follow Path" << endl;
        cout << " e           : Enter new current pose: " << endl;
        cout << " t           : Enter new target for path Planning: " << endl;
        cout << " n           : Network communication mode" << endl;
        cout << " x   : Quit" << endl;
    }

    /*if (!mrpt::system::os::kbhit())
    {
        robot.doProcess();
        CGenericSensor::TListObservations dummy;
        robot.getObservations(dummy); // Empty the list
        mrpt::system::sleep(20);
        continue;
    }*/

    char c;

    // ===== Update odometry
    CPose2D      odo;
    double       v,w;
    int64_t      left_ticks, right_ticks;
    bool         pollLRF = true;
    cout<<"getting odo"<<endl;
    thrPar.isMoving.set(true);
    while(thrPar.gettingLRF.get());
    //{
    //    pollLRF = returnGettingLRF(thrPar);
    //    cout<<pollLRF<<endl;
    //    sleep(1000);

    //}

    cout<<"OK"<<endl;
    getOdometry( odo, odo_fd, thrPar );
    thrPar.isMoving.set(false);
    //printf("***x = %d, y = %d, phi = %d\n",odo.x(),odo.y(),odo.phi());

```

```

        fixOdometry( odo, thrPar.odometryOffset.get() );
        thrPar.currentOdo.set(odo);
        //printf("***x = %d, y = %d, phi = %d\n",odo.x(),odo.y(),odo.phi());
        cout << "Odometry: " << odo << " v: " << v << " w: " << RAD2DEG(w)
<< " left: " << left_ticks << " right: " << right_ticks << endl;
        counter = 0;
// ===== End update odometry

        c = mrpt::system::os::getch();

        show_menu=true;

        if (c=='x') break; /* quit */

        if (c=='w' || c=='s') /* increase or decrease current linear velocity */
        {
            thrPar.goForward.set(true);
            thrPar.goRight.set(false);
            initial_front_wall = thrPar.front_wall.get();
            sleep(1000);

            if (c=='w') cur_v = 1;
            if (c=='s') cur_v = -1;

            setVelocities( cur_v, 0, thrPar);
            sleep(1000);
            //setVelocities( 0, 0, thrPar);

        }

        if (c=='a' || c=='d') /* increase or decrease current angular velocity */
        {
            //if (c=='a') cur_w = 1;
            //if (c=='d') cur_w = 1;
            thrPar.goForward.set(false);
            thrPar.goRight.set(true);
            if (c=='a') cur_w = 1;
            if (c=='d') cur_w = 1;

            setVelocities( 0, cur_w, thrPar );
            //setVelocities( cur_v, 0, thrPar);
            sleep(1000);
            //setVelocities( 0, 0, thrPar);

        }

        if (c==' ') /* stop, set current linear and anular velocities to 0 */
        {
            cur_v = 0;
            cur_w = 0;
            thrPar.goForward.set(false);
            thrPar.goRight.set(false);
            setVelocities( cur_v, cur_w, thrPar );

```

```

}
/*

if (counter >= 3) {
    c='o';
} else {
    counter += 1;
}

if (c=='o') // get current odometry reading
{
    CPose2D        odo;
    double         v,w;
    int64_t        left_ticks, right_ticks;
    bool           pollLRF = true;
    cout<<"getting odo"<<endl;
    thrPar.isMoving.set(true);
    while(thrPar.gettingLRF.get());
    //{
    //    pollLRF = returnGettingLRF(thrPar);
    //    cout<<pollLRF<<endl;
    //    sleep(1000);

    //}

    cout<<"OK"<<endl;
    getOdometry( odo, odo_fd, thrPar );
    thrPar.isMoving.set(false);
    printf("***x = %d, y = %d, phi = %d\n",odo.x(),odo.y(),odo.phi());
    fixOdometry( odo, thrPar.odometryOffset.get() );
    thrPar.currentOdo.set(odo);
    printf("***x = %d, y = %d, phi = %d\n",odo.x(),odo.y(),odo.phi());
    cout << "Odometry: " << odo << " v: " << v << " w: " << RAD2DEG(w)
<< " left: " << left_ticks << " right: " << right_ticks << endl;
    counter = 0;
}
*/

if (c=='p') /* get current bumper readings */
{
    vector_bool bumps;
    robot.getBumpers(bumps);
    cout << "Bumpers: " << bumps << endl;
}

if (c=='n' || c=='N') /* get current sonar readings */
{
    CObservationRange obs;
    bool thereis;
    robot.getSonarsReadings(thereis,obs);

    if (!thereis)
    {

```

```

        cout << "Sonars: NO" << endl;
    }
    else
    {
        adjustCObservationRangeSonarPose( obs );
        displaySonars(thrPar, obs);
        cout << "Sonars: ";
        for (CObservationRange::const_iterator
i=obs.sensedData.begin();i!=obs.sensedData.end();++i)
            cout << i->sensedDistance << " ";
        cout << endl;
    }
}

if (c=='b') /* get current battery charge */
{
    double bat;
    robot.getBatteryCharge(bat);
    cout << "Battery: " << bat << endl;
}

if (c=='e') /* enter current pose */
{
    double newX, newY, newPhi;
    cout << "Input the current x location: ";
    cin >> newX;    cin.clear();
    cout << "Input the current y location: ";
    cin >> newY;    cin.clear();
    cout << "Input the current phi: ";
    cin >> newPhi;  cin.clear();

    CPose2D startOdo;
    getOdometry( startOdo, odo_fd, thrPar);
    startOdo.x(0);
    startOdo.y(0);
    //CPose2D startOdo(newX,newY,DEG2RAD(newPhi));
    CPose2D newOffset(newX,newY,DEG2RAD(newPhi) - startOdo.phi()
);

    thrPar.odometryOffset.set(newOffset);
    robot.changeOdometry(startOdo);
    CPose2D      odo;
    double      v,w;
    int64_t     left_ticks, right_ticks;
    double      phi;
    //getOdometryFull( odo, v, w, left_ticks, right_ticks );

    fixOdometry( odo, thrPar.odometryOffset.get() );
    thrPar.currentOdo.set(odo);
    cout << " New odometry has been set! " << odo << endl;
    thrPar.pdfResetDeterministic.set(true);
}
}

```



```

if (c=="T") // Turn to target pose
{
    CPose2D      odo;
    /* calculate required phi to next point */
    double  phi  =  atan2 ( (thrPar.targetOdo.get().y() - (thrPar.targetOdo.get().x() ) ) ,
(thrPar.targetOdo.get().x() ) );

    turn(45.0*M_PI/180,thrPar);
    cout << "phi: " << RAD2DEG(phi);
    getOdometry( odo, odo_fd, thrPar );
    fixOdometry( odo, thrPar.odometryOffset.get() );
    thrPar.currentOdo.set(odo);
}

if (c=='t') /* enter target pose */
{
    double newX, newY;
    cout << "Input the target x location: ";
    cin >> newX;    cin.clear();
    cout << "Input the target y location: ";
    cin >> newY;    cin.clear();

    target.x(newX);
    target.y(newY);
    thrPar.targetOdo.set(target);
    thrPar.targetOdo.set(target);

    cout << " New target has been set! " << target << endl;
}

if (c=='P') /* calculate a path and travel there using the smoothDrive function */
{
    std::deque<poses::TPoint2D>      thePath;
    CPose2D      odo;
    double      v,w;
    int64_t  left_ticks, right_ticks;
    double      phi;
    getOdometry( odo, odo_fd, thrPar );
    fixOdometry( odo, thrPar.odometryOffset.get() );

    CPoint2D origin( odo.x(), odo.y() );

    if (PathPlanning( thePath, origin, target ) == 1)
    {
        thrPar.pdfResetDeterministic.set(true);
        thrPar.thePath.set(thePath);
        thrPar.displayNewPath.set(true);
        cout << "found a Path..." << endl;
        smoothDrive(robot, thePath, thrPar );
        cout << "at target..." << endl;
    } /* end path following */
}

```

```

        if (c=='n')
        {
                clientCommunication();
        }

}

/*join threads */
cout << "Waiting for grabbing thread to exit...\n";
thrPar.quit = true;
//mrpt::system::joinThread(pdfHandle);
mrpt::system::joinThread(thHandle);
//mrpt::system::joinThread(wallDetectHandle);
mrpt::system::joinThread(displayHandle);
cout << "threads ended!\n";
}
catch(std::exception &e)
{
        cerr << e.what() << endl;
        return -1;
}
return 0;
}

```