Tampere University

Tarun Reddy Devalla

SHARED CONTROL OF MOBILE ROBOTS USING MODEL PREDICTIVE CONTROL

Master of Science Thesis
Faculty of Engineering and Natural Sciences
Examiners: Associate Professor Reza Ghabcheloo
Associate Professor Roel Pieters
October 2020

ABSTRACT

Tarun Reddy Devalla: Shared Control of Mobile Robots using Model Predictive Control
Master of Science Thesis
Tampere University
Automation Engineering, Factory Automation and Robotics
October 2020

With the world constantly driving towards attaining complete autonomy, there is still a major question of safety when it comes to trusting a machine completely. Autonomous systems of today also do not have the ability to perform flawlessly in an environment that is cluttered and unstructured. This calls for the need of having a human operate the machine at all times either remotely via tele-operation methods or by being physically present alongside the machine. With tele-operation of remote systems, the cognitive load required from the human operator is high, while also the perception of the remote systems environment is low. This can cause many undesirable human errors causing damage to machinery. For example, tele-operating a forestry machine in a forest can be a very daunting task as there will be many trees and not all trees around the machine can be seen by the operator during remote tele-operation. With this in context, a few industries and sectors have now largely started research with using shared control methodologies to aid their machine in tele-operation tasks.

This thesis proposes a shared control methodology to provide a certain level of autonomy to the machine while still allowing the human operator to always be in control. The proposed methodology uses a Model predictive controller as the base controller to control the robot and perform obstacle avoidance tasks. The robot considered for implementation is a differential drive mobile robot, in specific the MiR 100 from Mobile Industrial Robots. The key motivation behind the thesis is to evaluate the performance of the shared control approach against a manual tele-operation task, to better understand the advantages and possible disadvantages of using a shared control strategy. The proposed strategy is implemented using the CasADi optimization toolbox on Matlab and tested through user testings. The results obtained from the user test prove that shared control can largely help in improving the safety of the system, but not so much with performance, at least not with the proposed methodology.

Keywords: Shared Control, Mixed Control, Human-Robot Interaction, HRI, Human in the Loop, Mobile Robot, Differential drive robot, Model Predictive Control, MPC, Receding Horizon Control, Obstacle Avoidance, teleoperation

PREFACE

This thesis is written as a part of the completion for my Master's Degree Program in Automation Engineering at Tampere University, under the department of Engineering and Natural Sciences. I would like to thank my thesis supervisor Prof. Reza Ghabcheloo for his mentoring and guidance throughout the thesis process, and for showing persistent faith in my capabilities. His trust has been a key motivation for accomplishing my masters degree. I would also like to extend my gratitude to my thesis examiner Prof. Roel Pieters for his valuable inputs and feedback. I thank all my fellow colleagues and peers for their support and advice during the course of my master's program.

I want to express my gratitude towards my parents, sisters and grand parents by dedicating this work to them. It is their unconditional support in all my decisions which has helped me in achieving my aspiration so far.

Finally, the note of thanks cannot end without mentioning my dear friends. They have been a significant part in molding me personally and in my career aspirations. On a special note, I would like to extend my gratitude to Pinaki Halale for her constant moral and emotional support towards the final phase of my thesis.

Tampere, 25th October 2020

Tarun Reddy Devalla

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PROGRAMS AND ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

2D	2-Dimensional
3D	3-Dimensional
CTRM	Contextual Task Recognition Module
DOF	Degrees of Freedom
d_{RO}	Distance between the robot and the obstacle
e.g.	Exempli gratia (For example Latin abbreviation)
Et al.	et alia
FRVF	Forbidden Region Virtual Fixture
GMM	Gaussian Mixture Model
GMR	Gaussian Mixture Regression
GVF	Guidance Virtual Fixture
J	Cost/Objective function MPC controller
k	Current time step in discrete time
LaTeX	a document preparation system for scientific writing
LOA	Levels of Autonomy
MCAM	Motion Command Arbitration Module
MDP	Markov Decision Process
MiR	Mobile industrial Robots
MPC	Model Predictive Control
N_O	Total number of obstacles
N	Prediction Horizon for the MPC controller
POMDP	Partially Observable Markov Decision Process
Q	3X3 Positive definite diagonal weighting matrix for the States
QMDP	Q-learning Markov Decision Process
R	2X2 Positive definite diagonal weighting matrix for the control inputs
RBF	Recursive Bayesian filter
r_O	Radius of the obstacle

r_R	Radius of the robot
TAU	Tampere University
θ_R	Robot heading
$\dot{\theta}$	Rate of angular change of the robot
θ_{ref}	Goal heading
θ_P	Heading of Human controlled Point
TUNI	Tampere Universities
u	Control variables for the MPC controller
URL	Uniform Resource Locator
v_{acc}	Limit on the increase of linear velocity
v_{dec}	Limit on the decrease of linear velocity
v	Linear velocity
w_{acc}	Limit on the increase of angular velocity
w_{dec}	Limit on the decrease of angular velocity
w	Angular velocity
x	Robot State
$\dot{x}$	Velocity component along x axis
x_g	Goal state
x_{ref}	X axis position of Robot goal
x_P	Pose of human controlled point
x_P	X axis position of Human controlled Point
p_{obs}	Obstacle position
x_{obs}	X axis position of obstacle
x_R	X axis position of Robot
$\dot{y}$	Velocity component along y axis
y_{ref}	Y axis position of Robot goal
y_P	Y axis position of Human controlled Point
y_{obs}	Y axis position of obstacle
y_R	Y axis position of Robot

1 INTRODUCTION

Today autonomous systems are peaking interests in quite many fields, with many disciplines seeking out autonomous solutions for automating processes thereby reducing human labour. This approach might be pretty straight forward and an achievable dream for fixed tasks in well-defined environments. But, when we take into consideration completely unstructured environments with dynamic changes, automated systems are still not preferred. The drawbacks of today are not restricted just to the environmental factors, but, also due to high complexity of some machinery that can make it difficult for complete automation. Tele-operation of such machines will also require highly skilled and trained operators [13].

There has been a rise in autonomous systems being deployed in various fields such as defense, aviation and automobile without having high predictability of the environment and a system where failure is completely acceptable [2]. At times such as this, humans are in complete control of the vehicle either on the machine or through tele-operation as they have the capability to perceive the environmental conditions well enough to control the machine. Even so this poses a few problems with tele-operation like, limited situational awareness of the environment and communications delays. This can lead to unexpected collisions in the environment and damage to the machinery [40].

Tele-operation can be a task that requires high cognitive abilities from the human operator demanding high levels of alertness at all times. This can become a very demanding job especially in environment with many dynamically moving objects. Tasks like search and rescue operations, require a human to be in control of the machine always, like in using tethered underwater rescue robots which is just one example. The tele-operation of the machine can prove to be a rather daunting task without the right system design and providing a high situational awareness to the user [29].

With more data being transferred to the user, to provide a good feel of the real system, the delays within the system can start drastically increasing as well. The user will, however, not have complete feel of driving a real machine and can become very comfortable with controlling the system at full throttle. When in fact, the system may at times not behave as expected, causing an undesired movement of the robot.

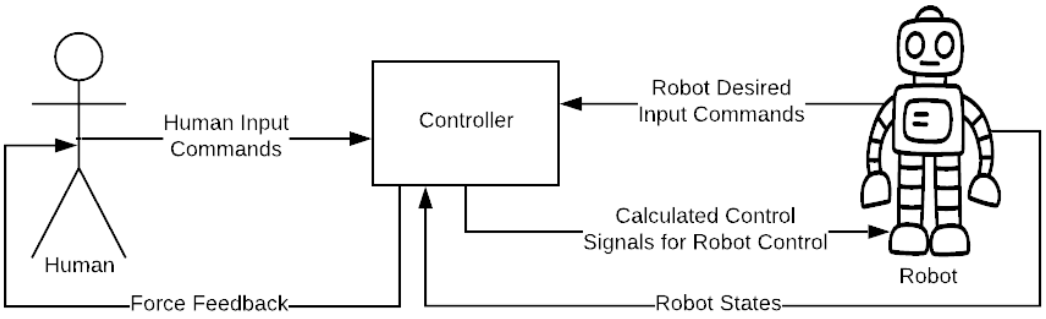


Figure 1.1. *A generalised representation of a Shared Control System.*

1.1 Motivation

Shared control can be seen as a solution bridging the gap between completely autonomous systems and completely manual systems by introducing an architecture that has the human always in the loop, while also giving certain level of autonomy to the machine. This might now raise another question, what is shared control? Till date, shared control does not have an outright definition. A shared control system can simply be recognised as any system within which task execution collectively depends on both the human input and input from an autonomous controller on the machine.

A generalised representation of a shared control system can be seen in Figure 1.1. The controller, takes as input, the desired control inputs from the human operator, the inputs calculated by the robot's on-board PC and the robot's current state. It uses these to calculate a desirable control value for moving/manipulating the robot. The controller can also give haptic force feedback to the operator to increase the operators perception on the task execution. The system may also have additional aids for the human to increase perception. For example, a video feed or visualization of the sensor data.

Shared control systems are by large increasing in popularity off late with 5% increase in year-by-year number of publications. There has also been a diversification to the different applications where shared control has been used, which include:

1. Automotive industry
2. Drone control and navigation
3. Robotic surgery
4. Brain Machine Interfaces (BMI) for prosthetic and wheelchair control
5. Mobile robots control and navigation

These are just a few of the prominent application areas where shared control is currently in use [2]. But on a closer look, we can conclude that most of these tasks still require the

human to be in the control loop simply because these tasks can be very time critical and the human will have a better judgement and outlook towards the situation when compared to using an autonomous system [39]. Then again, this raises the question of safety and efficiency of using a manual tele-operation method.

This brings systems with shared control strategies into play, to enable more dynamic and safe tele-operation of mobile machines and remote robotic systems. Shared control can be implemented in various different methods as the concept is rather very broad. A few different methodologies will be discussed later in the next chapter to give a good idea of how a shared control approach can be adopted for the task at hand.

1.2 Research Questions

As mentioned above, shared control can be used for various applications. But to evaluate a shared control architecture, a few questions need to be tended to.

1. How to include Human in the Loop control for shared control?
2. How can having a shared control approach help with more efficient robot control in terms of maneuverability?
3. Can the use of a shared control approach help with increasing the safety of robot during tele-operation?

1.3 Objectives

The key objective of the thesis is, to develop a shared control architecture to enable a human-in-the-loop control scheme using a 3D simulation environment and a mobile robot. A mobile robot can be either a small indoor mobile robot, like the MiR100 or a much larger mobile machine, like a hydraulic wheel loader. For this thesis we use a non-holonomic mobile robot, MiR 100, to test the implementation. But, the method should be portable to any mobile machine using the appropriate system model.

To do so a suitable test environment needs to created to test the shared control system that will be implemented. The main robot controller will be controlled using Model Predictive Control (MPC) approach and the user will provide inputs to the system using a joystick of 2 DOF, as the current robot has only 2 controllable Degree of Freedom. After implementation of the shared control architecture, the system is to be tested with users of different skill level and technical backgrounds to achieve an unbiased analysis of the system.

1.4 Thesis Structure

The document has a total of 6 chapters, with the first being the introduction of the thesis, giving an idea to shared control systems and scope of this thesis.

Chapter 2 gives a theoretical background to the different shared control implementations, along with a short literature review on MPC approaches.

Chapter 3 presents the thesis methodology and implementation which will include the kinematic modelling of the mobile robot, the MPC formulation, introduction to the simulation environment and the proposed shared control architecture.

Chapter 4 focuses on presenting the approach taken for performing the user tests of the implemented shared control methodology mentioned in chapter 3. This chapter also provides the results from the user testing and along with a comparison of performance between a shared control approach against a conventional tele-operation approach.

Chapter 5 provides a comprehensive analysis based on the results from user testing highlighting the shortcomings of the implemented shared control architecture.

Finally, Chapter 6 concludes the works of this thesis, along with future works to further enhance and develop the research on this topic.

2 LITERATURE REVIEW

As discussed in chapter 1 shared control doesn't have a discrete definition and this can make this a rather widespread topic with various different applications and implementations to it. To grasp a clear and concise understanding of the topic, a comprehensive literature study of some of the methodologies will be presented below. Shared control in a generalised format can be seen as a collective implementation of the following three parts [24]:

1. Intent Detection
2. Arbitration
3. Communication

We will have more focus towards the arbitration topic of shared control with a brief overview on communication. After this we will look into a brief study on the different application and implementation of Model Predictive Control as this will serve as the base robot controller in this thesis.

2.1 Shared Control: Intention and Arbitration

Arbitration in context of shared control refers to how and when control is shared between the autonomous system and the human operator. There are numerous methods by which the human operator can interact with the autonomous system and this can be seen from the Levels of Autonomy (LOA) [12, 16]. There are a total of 10 LOA, which range from 1-being complete manual operation to 10-being fully autonomous operation. The taxonomy of LOA and their descriptions can be seen in table 2.1. Shared control strategies can be considered as a system that adopts any LOA between and including levels 3 and 6. A more apropos argument would be that shared control method adopts the 4^{th} LOA.

2.1.1 Virtual Fixtures

Virtual fixtures are the most common and preferred method of implementing shared control in the field of robot assisted surgeries and mobile robotics. Virtual fixtures, also referred to as Virtual constraints/active constraints can be viewed as attractive or repulsive

No.	LOA	Description
1	Manual	Computer/autonomous systems offers no assistance. Meaning the human operator performs all the tasks.
2	Action Support	The autonomous system assists the human operator in performing certain tasks by providing a complete set of action. Some human input is required to help the autonomous system in deciding which action to perform.
3	Batch Processing	Human generates/selects the options to be performed by the autonomous system thereby narrowing down the complexity of autonomous system to only physical implementation of the selected tasks.
4	Sharing Control	At this level, the autonomous system does not have any decision making capabilities but rather works based on human generated control strategies and options. Based on the generated strategies, the task of the autonomous system is to apply it by modifying it at any time the proposed strategy seems to be unsafe.
5	Decision Support	The computer provides a list of decision options to human to choose from or the human can also generate his/her own decision options for the system at any time. After a certain option has been chosen, the autonomous system can now implement it.
6	Blended Decision Making	This is pretty similar to the previous LOA, but, rather than asking for human decision, the autonomous systems carries out its own generated strategy until the human intervenes with a new strategy.
7	Rigid System	The autonomous system presents only a limited set number of options to the human to select from. Unlike the previous two LOAs, the human cannot generate his/her own options/strategies for the system to perform.
8	Automatic Decision Making	The autonomous system generates a set of options/strategies and picks the best out of them to implement. In this LOA the set of options/strategies can be generated either by, the human or the autonomous system.
9	Supervisory Control	The computer performs actions and tasks on it's own autonomously while the user is constantly or periodically monitoring the system. The user can at anytime intervene if required, changing the system temporarily to a decision support LOA.
10	Full Automation	The computer is in control of all tasks and actions, without any need for human intervention.

Table 2.1. *The LOA taxonomy for dynamic systems [12, 16]*

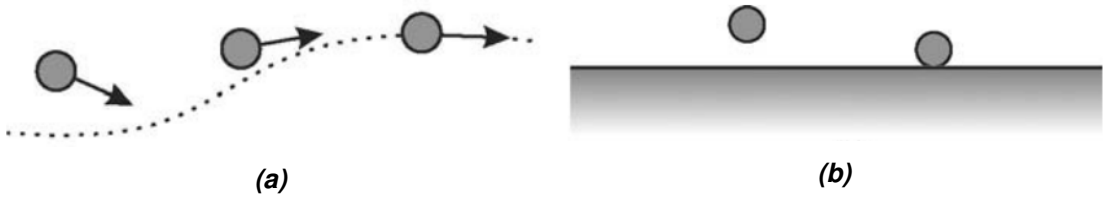


Figure 2.1. *(a) GVF assisting a robot to follow a path (dotted line), (b) FRVF preventing robots from entering a forbidden region(grey shaded region) [3].*

forces imposed onto the environment. There are different methods to implement the virtual fixtures in a task space. In a paper presented by Abbot J.J. et al. [3] they categorize Haptic virtual fixtures to be of two types:

1. Guidance Virtual Fixtures (GVFs)
2. Forbidden Region Virtual Fixtures (FRVFs)

An example representation of these virtual fixtures can be seen in figure 2.1

Guidance virtual fixtures are regions or paths in the environment that help the user towards task completion. These can be either pre-programmed for known environments or can be drawn onto a 2D representation of the robot perception by using human interaction, which is then projected onto the actual task space in 3D [34]. When working with unknown task spaces the latter method might be preferable but comes with the shortcoming of time used for defining the intended paths. Another method as proposed by J. Yan et al. [44] takes a different approach where they define guidance regions within which the robot can freely operate. They can be of different shapes/types depending on the task to be performed. A few of these include,

1. Line guides: The robot must follow the line at all times.
2. Plane guides: The robot motion is constrained to be within a defined plane region at all times. The plane can be of any shape. For example, triangle, trapezoid and rectangle.
3. Solid-type guides: The robot motion is confined to a 3D region in the task environment that may take the shape of a solid object like a cone, cylinder or cuboid.

A visual representation of the different types can be seen in the figure 2.2. Here the line guides act as GVFs while all other types of virtual fixtures can be seen as FRVFs.

Forbidden Regions are defined to prevent the robot from moving into undesirable location in the environment. This type of virtual fixtures can be of large for applications of hydraulic machinery where the vision of the human operator is limited. This could help in preventing unwanted actions knowingly or unknowingly that could happen during operation of the machine.

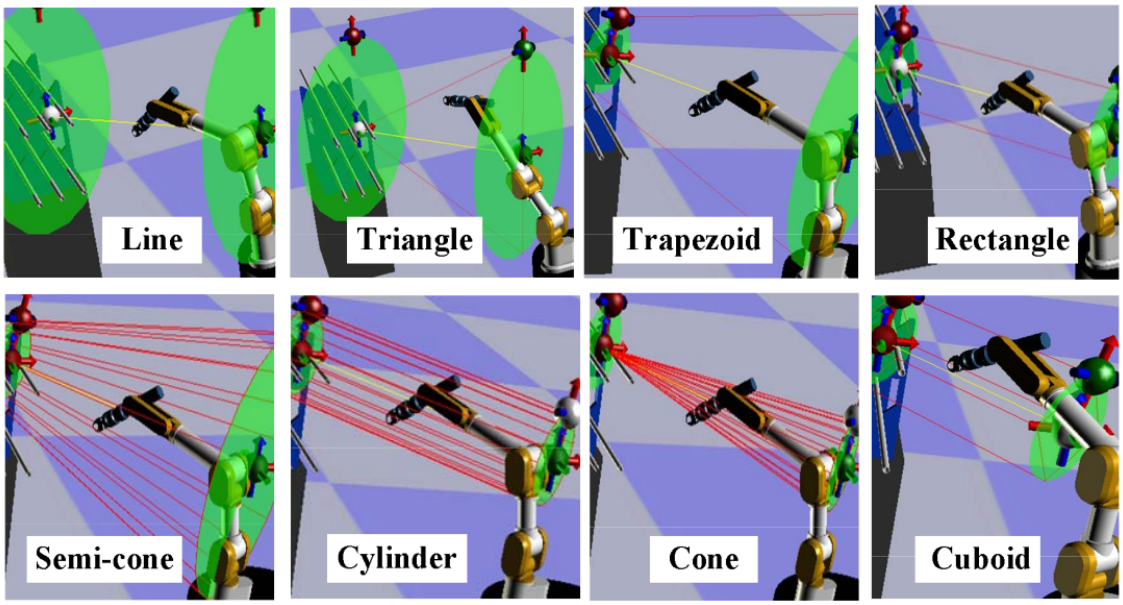


Figure 2.2. *Types of virtual guides [44].*

Another method to define these virtual fixtures is by the method of Programming by Demonstration [46]. In this approach the virtual fixtures are thought to the controller by demonstration. The controller then uses a weighted average of human confidence and robot confidence to better estimate the desired action to be performed. Another machine learning based virtual fixtures was discussed in the paper by D. Aarno et al. “Adaptive Virtual Fixtures for Machine-Assisted Tele-operation Tasks” [1]. In this paper the authors divide the complete task into several subtasks or goals and use a training model with virtual fixtures. This process was carried out using Hidden Markov models. The algorithm later learns to adapt to changes in the environment. The advantages that the two machine learning approaches bring about are, it reduces the need for programming the virtual fixtures offline as the task can become complicated and time consuming when the environment is frequently changing.

Virtual fixtures can be generally implemented with robotics systems as an impedance or admittance type control structure, as these are already common and widely used. Impedance control can not only be used for traditional control, but, also as a control algorithm for shared control applications too. A method proposed by P. Nadrag et al. [30] uses impedance control to control a mobile robot using a haptic device. They perform a task of obstacle detection and avoidance by measuring the distance to obstacles and thereby providing the required force feedback to the human operator. This method can also be applied to applications with Forbidden Region Virtual Fixtures [3].

The method above uses only an impedance controller which lets the user move freely in the space as long there are not any obstacles in the environment. This does not provide any task awareness to the user. This can be overcome by using a combination of both

Impedance and admittance type controllers.

N. Yu et al. [45] have proposed an algorithm which fuses impedance values calculated from robot dynamics with admittance values from user input to provide a realistic force feedback that the user feels. This force guides the user in obstacle avoidance rather than just simple force/vibro-tactile feedback that gives only an awareness to the user. Abbot. J. J et al. [3] have also proposed an admittance type controller that can be used with Guidance Virtual Fixtures for tele-manipulation tasks.

2.1.2 Mode Switching

The above methods although do not mention anything about the method of tele-operating the robots. High DOF control modules can at times seem to be very expensive and that is alright for only research purposes. At most times there might not be a lot of funds to spare on only a controller and this also posed a problem for implementation in this thesis, where simple 3 DOF Haptic Devices can cost around a thousand euros. Higher DOF machines can also be very hard to learn to operate and control for someone who does not know the system well.

A probable to solution to this problem was proposed by Srinivasa et al in the paper “Assistive Tele-operation of Robot Arms via Automatic Time-Optimal Mode Switching” [13]. They propose a method of time optimal mode-switching to help in change between different operating modes of the device. This may raise a question, why not manually change modes during operation? That can of course be done however that increases the time and cognitive load on the operator and that affects the efficiency of performing the task. Using a simple time-optimal mode switching method, the Authors were able to change between different modes while also delivering user satisfaction [13]. With such a model, even a complex arm with 6 DOF can eventually be controlled by a device with only 3 DOF, like a simple Joystick effortlessly. An experiment conducted on this proved that with manually switching the modes there was 17.4% time increase in task execution compared to a model with time-optimal mode-switching.

2.1.3 Sliding Autonomy

As mentioned earlier in section 2.1 there are various LOA that a system can adopt. An interesting shared control approach is to have a system that can dynamically change/-control the LOA the system is currently working on, seamlessly during run-time. A very good example to such a system can be seen in air crafts. The pilot can set the plane to complete Autopilot or can choose to control the heading or height or speed manually. This methodology is generally referred to as sliding autonomy and can be obtained using different methods such as Standard Dial Approach, Hierarchical Approach or Policy

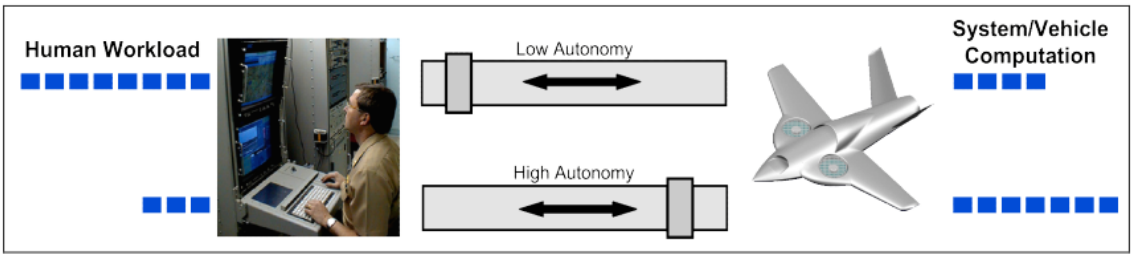


Figure 2.3. *A representation of sliding dial approach for balancing workload between operator and autonomous system [28].*

Based approach [28]. An simple interpretation of the sliding dial approach can be seen in the figure 2.3. The paper compares the 3 methods mentioned and concludes that Policy Based approach for sliding autonomy can be the most beneficial in terms of task complexity. But this brings other implementation difficulties like ease of use by the user and design of an intuitive control interface.

Apart from these approaches, a system can also be completely autonomous until it comes to a deadlock, a situation from which the autonomous system cannot recover without assistance from the human operator. In case of deadlock, autonomy can be completely switched off and operator can tele-operate the robot to recovery point/state from which it can resume tasks autonomously. A problem with such a system is that some tasks require the user to know exactly how the robot got to that position or the user must be able to at least determine the cause [36]. This is required so that the user can recover the robot to a position such that the robot will not repeat the mistakes again. This situational awareness can be brought forth to the operator using recordings from the robot just before it went into a deadlock. This is much more feasible than having to do a continuous live stream over the network. Even with current trends of wireless technology, large throughput of data transmission of high bandwidth is not feasible and might sometimes cause hindrance to transmission of more crucial data.

2.1.4 Policy Blended/Probabilistic Approaches

A virtuous approach to having a human-in-the-loop control architecture is to adapt to a blended control structure wherein at all times the user inputs and the autonomous controller inputs are used together to come to a final control decision. A common methodology of blending the user inputs is using an α value that gives the blending factor between the user input and the control input from the autonomous system [39].

Policy blending can also be done by having a set of known tasks to be performed and effectively estimating the confidence levels of the automation system progressively based on the inputs from human operator. This approach was adapted in a paper by Dragan et. al [9] and a simple control flow of a policy blended approach can be seen in figure 2.4.

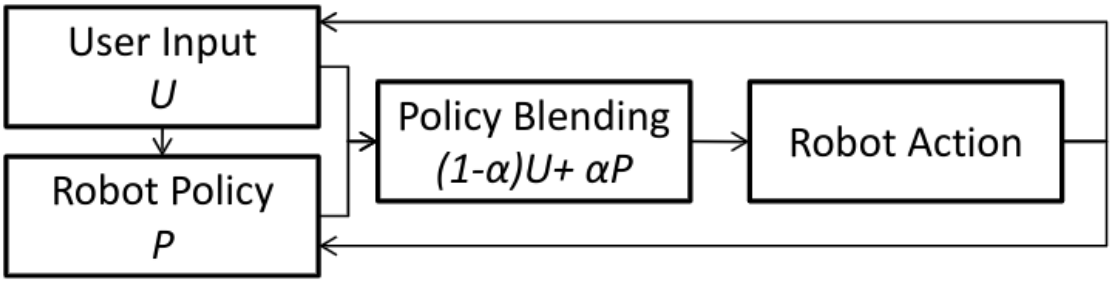


Figure 2.4. *Control Flow of a Policy Blended Shared Control Structure [9].*

They have two scenarios predefined. 1. Grasp Bottle(Easy) and 2. Grasp Box(Hard). These two scenarios in turn give rise to 4 tasks that can happen: Easy-Right, Easy-Wrong, Hard-Right and Hard-Wrong [9]. The authors also define the arbitration function to be dependent on the confidence of the robot in the current policy and describes the confidence to be inversely proportional the distance to the object.

Another interesting approach is using hindsight optimization, where the system uses a Markov Decision Process (MDP), to determine the most probable goal from user inputs and move more efficiently towards the goal based on the input values. In a paper by S. Javdani et al. [15] they approximate the optimal action using QMDPs which are a combination of MDPs and Partially observable Markov Decision Processes (POMDPs). The experimental setup consists of known object positions in task space and assumes each object can have any grasp pose. The system uses the user input to estimate the best possible path to follow and also which object is desired by the user to be grasped using directional guidance from user. It can be better understood from the illustration extracted from the paper shown in figure 2.5. We can see initially all object have a similar probability of being the final goal object to be grasped, but as human input is given the robot can better estimate which object is to be grasped and proceed with the task. This implementation can be advantageous in situations of having to decide between multiple goals to choose from.

Another probabilistic approach is discussed by Liang et al.[22] where the authors use a Contact model to define the robot dynamics and an intent recognition algorithm that is used to detect the relative distance from end-effector and target location and act accordingly. The closer the user moves the end-effector towards an object, the probability increases. With a high enough probability, the system then performs task without need for human input. In the paper the authors compare the time consumption for direct control with and without feedback against shared control with and without feedback.

With all these methods a key role of the shared control system lies in human intent detection. This let's the human decide on the best action based on inputs from the human operator. A method proposed by M. Gao et al. [11] uses a 2-part system that blends human intention and robot inputs based on estimation. The Contextual Task Recognition Mod-

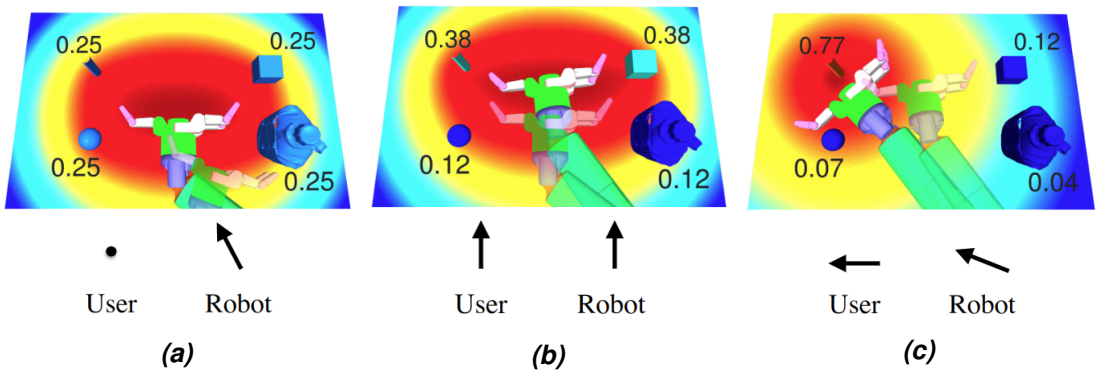


Figure 2.5. *Estimation of goal probabilities and value function with effect of human input on the system [15].*

ule (CTRM) uses Recursive Bayesian Filters (RBF), which comprise Gaussian Mixture Models (GMM) and Gaussian Mixture Regression (GMR) do determine the user intended tasks. The Latter part of the system, motion command arbitration module (MCAM) uses the estimation from the CTRM and uses the robot input to get a blended motion command[11].

2.2 Shared Control: Communication

The above-mentioned methods of arbitration focus mainly on how the control is shared between the user and robot. While robot and human work in unison, there needs to be a way to let the human operator know or comprehend the properties of the physical environment. This can be done by either simple vibro-tactile feedback units, visual aids, audio feedback or force feedback. Let us take an example of a car having lane assist. The system has certain level of control, but if the system does not respond actively to the drivers input and rather works in autonomous mode, the driver in this condition will gain misconception of the situation. While, if the same system has a force feedback steering wheel, the driver is informed of the system intentions and behaviour through the force feedback steering, that helps to stay on the lane[25, 41].

We have discussed previously in section 2.1.1 about virtual fixtures. These virtual fixtures can also be considered to be haptic virtual fixtures which provide a force feedback to the user. This allows the user to gain a perception of the robot in real environment and perform the task even with minimal visual aids using only the guiding forces provided by the virtual fixtures. A paper presented by M. E. Konts et al.[20] makes use of this concept for tele-operation task of forklifts. From the experiments conducted in that paper, users were able to perform the tasks with more ease and confidence while they had force feedback compared to a system without force feedback. A similar yet interesting implementation of the use of force feedback was in the training toddlers to steer[4]. The authors provide a 2-Dimensional force feedback through a joystick to the toddlers to help

train them in steering tasks. The results of this experiment were surprisingly positive, as they were able to learn to follow lines.

Another mode of communication is through visual aids to the user. These can be really helpful when coupled along with force feedback modalities. For example, when the operator can visually see the virtual fixtures, it would be more helpful during operation. Also as discussed previously in Section 2.1.3, the use of visual aids can pose to be really helpful but can in turn be costly in terms of data that needs be transmitted over the wireless network.

In an experimental study presented by F. Mars et al.[26], the authors test various levels of Haptic Shared control. The results of this experiment showed that the need for visual aids reduced with a system of comprehensive and heavy force feedback. With this being said, our focus can shift more towards haptic communication. This is because while developing a tele-operation we must also consider the time taken for data transmission. When we include video transmission into pipeline, it will require a lot data bandwidth for effective lossless transmission.

2.3 A Brief Summary of Shared Control Methodologies

The mentioned shared control strategies help in providing a good idea on how a shared control system can be implemented and how it can seem to be beneficial over using a traditional tele-operation method. Virtual fixtures are by far the most common approach taken towards a shared control strategy to help aid in tele-operation tasks. This is due to the ease of formulating a system to use virtual fixtures for maneuverability tasks. But since this method still has the human doing most work, it can still seem to be not quite effective.

For this reason, policy blended/probabilistic approaches can have a vital impact on improving task efficiency while using a shared control approach to perform tasks. This methodology is again quite a broad classification of the various different method of implementation that can be adopted. This includes, policy blending, reinforcement learning, deep neural networks and other probabilistic methodologies like using MDPs, Gaussian process models and Model Predictive Control(MPC). Using an MPC can seem to be beneficial as it can include machine learning methodologies into the defined model, helping in better problem formulation. MPC's also allow for different implementation strategies for a same problem, while also allowing different ways to include human control into the model. This is a key reason MPC will used in this thesis. The next section takes a short look into an introduction to MPC and a few methodologies.

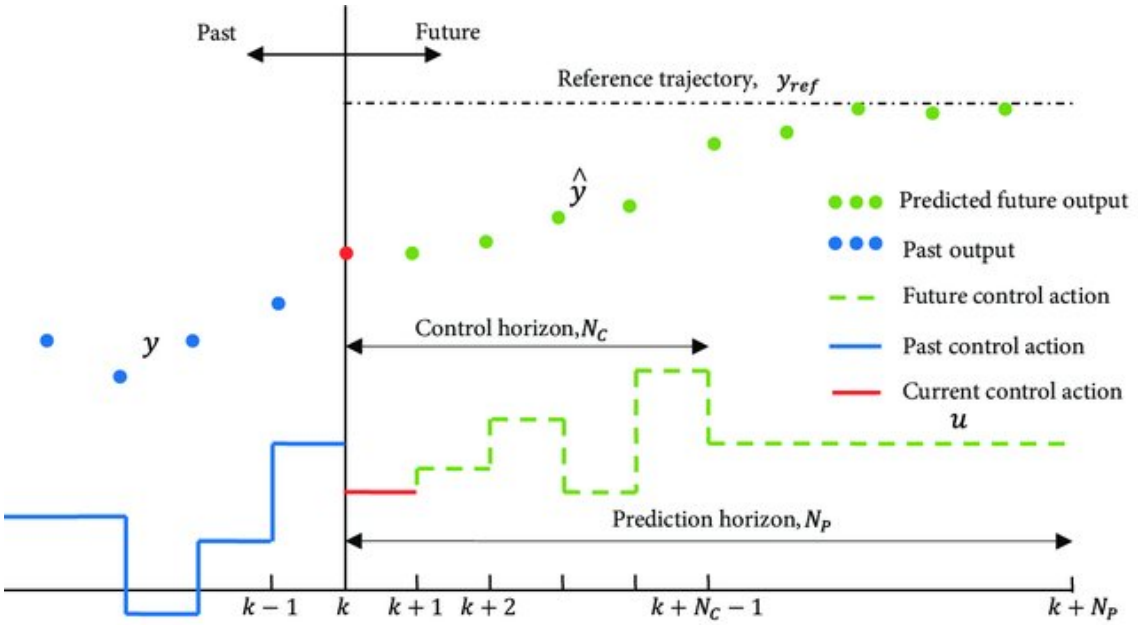


Figure 2.6. *Model Predictive Control Strategy [7]*

2.4 Model Predictive Control

In this thesis, a shared control approach will be developed using Model Predictive Control (MPC). Model Predictive Control also referred to as Receding Horizon Control has been gaining large importance and use over the course of almost 4 decades . MPC initially developed for the oil industry by Shell Oil in the 1970's [14]. But MPC has now found it's way into various different fields and applications that include control of aerial vehicles, robot locomotion[10] and power electronics[42] to name a few. This large increase in usage of MPC is largely due to the reason that it is possible to easily develop Multiple Input Multiple Output system including system constraints. More traditional control structures do not allow this. MPC can also be considered to be an Optimal Control algorithm repeating at each sampling interval to get the best/optimal solution [31].

To better understand the general outline of how an MPC strategy works let us take up and example of trajectory tracking with a given reference trajectory as seen in figure 2.6. In the figure y is the current output of the system and u is the control action on the system. MPC calculates the predicted control actions for the system over a Prediction Horizon and Control Horizon at each time step and applies only the first control action from the predictions to the system at each time step.

An MPC has 3 main parts to it irrespective of the application to be tended to[31].

1. Model of the system
2. Objective Function/Cost Function
3. Predictive Control Law to minimise the cost function

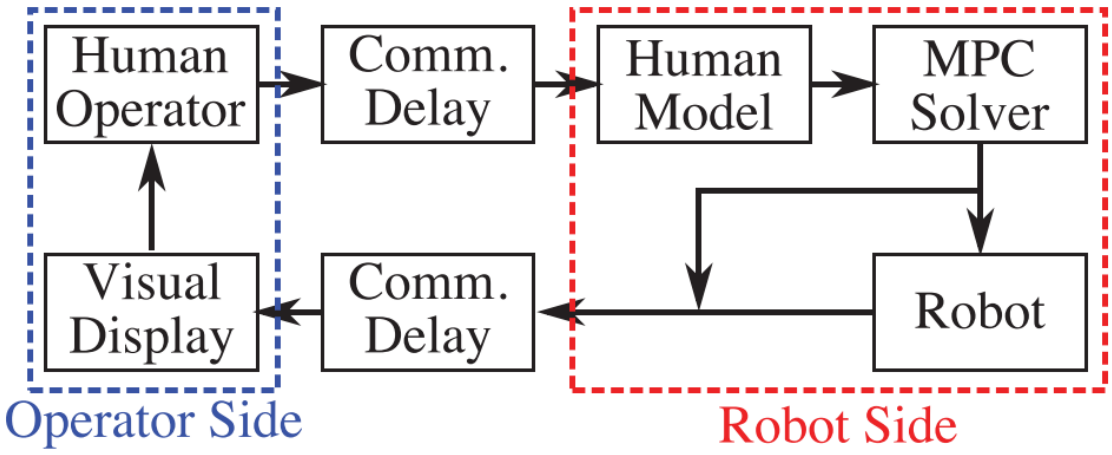


Figure 2.7. *The system model presented in [40] for an MPC based shared control approach.*

The system model of an MPC algorithm is analogous to the system that is being controlled by the MPC. Even so, a system model for a specific vehicle can be modelled using different methodologies. For example, a mobile robot can be modelled using either only the system kinematics, or by including the dynamic model of the system as well as seen in the researches presented in [5, 18, 21, 39, 45].

Next comes the formulation of the cost function and constraints that will be used for the MPC approach. This is wholly dependant on the task to be accomplished and the robustness of the system design. The cost functions are largely formulated prior to task execution based on the users understanding of the system and the environmental capabilities. Given a proper MPC formulation the system behaviour will be as close to the intended behaviour and the cost function and constraints would fit the defined use case. But sometimes this might not be case, and the controller behaviour can become undesirable. This problem can be overcome by using deep-learning or Q-learning approaches using neural networks to comprehensively update the cost functions and constraints of the MPC using prior test data, while also learning on the go[10, 32, 33].

Model predictive control approaches can also be used in shared control methodologies to have a human in the loop control architecture. There has been research on different methodologies on how to include the human inputs into the MPC formulation. One research suggests the use of MPC to generate a blending factor α. In this method, the user and autonomous system both give to the system a desired input value. The autonomous systems values are based on an obstacle avoidance scheme built using potential fields. The α helps in continuously changing the level of control between the robot and the human to enable a safe control structure[39]. One methodology was proposed for use in commercial road vehicles for effective lane keeping assistance through shared steering control. This model uses a smoothly shifting control authority model based on a confi-

dence estimate of the user model to predict human errors and correct the system, while also providing the user with constant haptic feedback of the systems intentions[38].

A method proposed by Storms Et. al.[40] closely relates to the methodology that will be solved as part of this thesis. In this paper the authors develop multiple shared control strategies with MPC as the base controller. They test shared control strategies that include only human control decisions, and also a method that takes into consideration the humans desired state manipulation of the robotic system to evaluate the performance criteria of different approaches including a communication delay on the system. One of the adopted shared control strategies, allows the user to control a point ahead of the robot to manipulate the robot steering accordingly. The authors use a robot model with constant linear velocity and allow manipulation of only the angular velocities of the robot. Hence, the point controlled by the robot only affects the angular velocity of the robot, thereby, steering the robot. The system model proposed in this paper can be seen in the figure 2.7.

3 METHODOLOGY AND IMPLEMENTATION

In the previous chapter we have taken a look into the different methodologies and implementations of shared control and MPC which were related mostly to mobile robotics applications, which is the main focus of this thesis. This chapter will now present the MPC formulation with the proposed shared control architecture for having a human-in-the-loop control architecture.

3.1 Shared Control Methodology

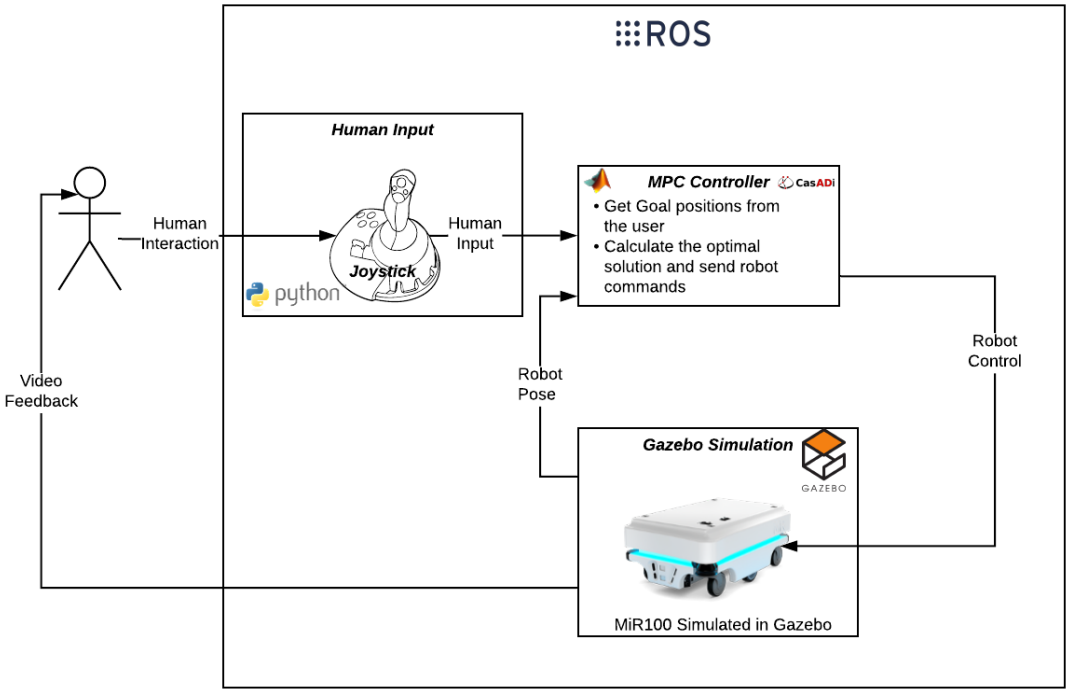


Figure 3.1. *Schematic view of the implemented shared control architecture with the different software/tools used at each block.*

For this thesis we consider a rather generic task of robot maneuverability to test the benefits of using a shared control approach over a traditional tele-operation approach. In the proposed methodology, we assume the robot has complete knowledge about the environmental constraints including obstacles, but the robot does not have any decision

making capabilities on the final goal that it must reach. This knowledge is input to the system perpetually by the human user. A simple representation of the shared control architecture carried out in this thesis can be seen in figure 3.1. As seen in illustration, the MPC algorithm is the heart of the implemented shared control architecture. The formulation of the MPC algorithm for the approach was carried out in 3 phases:

1. MPC for direct robot control with a fixed final goal in an environment without obstacles, explained further in section 3.1.2.
2. MPC for direct robot control with a fixed final goal in an environment with obstacles to test obstacle avoidance, explained further in section 3.1.3
3. Including human input to continuously update the desired goal for the robot with obstacle avoidance, explained further in section 3.1.4

3.1.1 Kinematic Modeling of a Differential Drive Mobile Robot

Figure 3.2. *MiR 100 Mobile Robot [27]*

The first part required for an MPC algorithm is the system model or plant model. In this thesis we will be using a Differential Drive Mobile Robot (DDMR), MiR-100 from Mobile industrial Robots (MiR)[27]. A picture of the robot in discussion can be seen in figure 3.2. A differential drive mobile robot has 2 individually controllable wheels on either side of the robot. The robot is controlled by controlling the linear velocity v and the angular velocity w of the robot. This makes the robot non-holonomic solely due to the fact that the robot can move linearly only in one axis (Forward$\leftrightarrow$Backward), and rotate about it's yaw axis restricting the robot to 2 DOF. Now we have the robots control input $u = \begin{bmatrix} v & w \end{bmatrix}^T$.

To formulate the kinematic model of the robot, consider the illustration in the figure 3.3. In

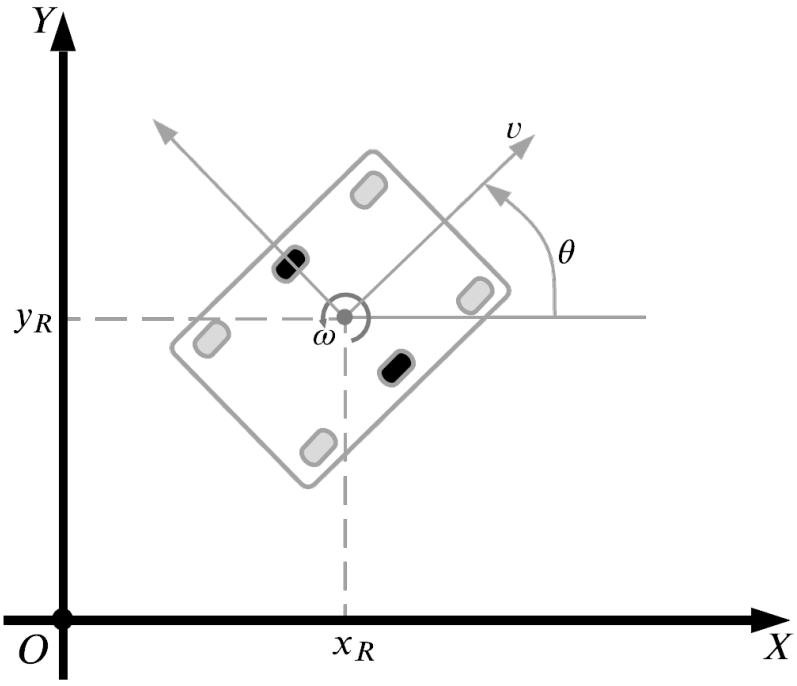


Figure 3.3. *2D representation of the MiR100*

the figure $\mathbf{O}$ is the origin of the world coordinate frame. The two individually controllable wheels of the robot are depicted with the black wheels, while the non-controllable castor wheels of the robot are shown in grey colour. The robot position is defined with respect to the world coordinate frame as x_R, y_R and θ_R giving the pose of the robot. Where x_R, y_R provide the position of the robot centre and θ_R provides the robot heading/orientation. This gives the state of the robot which can be denoted by x.

$$x = \begin{bmatrix} x_R \\ y_R \\ \theta_R \end{bmatrix} \tag{3.1}$$

The pose of the robot can be controlled by controlling the linear velocity and the angular velocity, v and w respectively. The linear velocity can be decomposed to the velocity along x and the velocity along y. This can be explained with the illustration as seen in figure 3.4. From the figure we can see it to be a right angled triangle and this will hold true for all cases of a 2D velocity vector like v. The velocity components $\dot{x}$ and $\dot{y}$ form the adjacent and opposite side of the triangle with v as the hypotenuse of the triangle. The angle θ can be seen as the angle made between the velocity vector v and velocity component $\dot{x}$. We can now derive the equations for the x and y velocity components

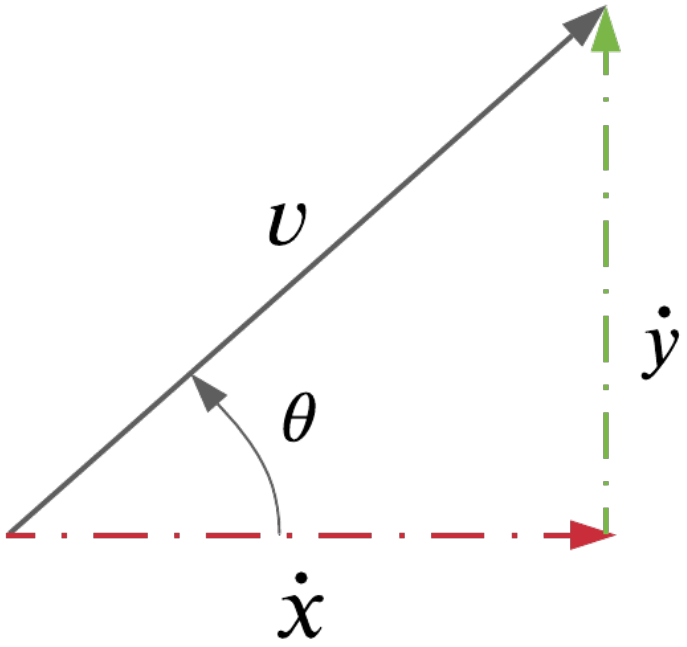


Figure 3.4. *Illustration of velocity vector v decomposition*

from the basic trigonometric equation of a triangle which are:

$$\cos\theta = \frac{\dot{x}}{v} \tag{3.2}$$

$$\sin\theta = \frac{\dot{y}}{v} \tag{3.3}$$

Rewriting equations 3.3 and 3.2 we get,

$$\dot{x} = v\cos\theta \tag{3.4}$$

$$\dot{y} = v\sin\theta \tag{3.5}$$

The angular velocity w directly affects the the rate of angular change of the robot $\dot{\theta}$.

$$\dot{\theta} = w \tag{3.6}$$

From the three equations we can now derive the kinematic model of the system

$$\begin{bmatrix} \dot{x}_R \\ \dot{y}_R \\ \dot{\theta}_R \end{bmatrix} = \begin{bmatrix} \cos\theta_R & 0 \\ \sin\theta_R & 0 \\ 0 & 1 \end{bmatrix} u \tag{3.7}$$

OR

$$\begin{bmatrix} \dot{x}_R \\ \dot{y}_R \\ \dot{\theta}_R \end{bmatrix} = \begin{bmatrix} v\cos\theta_R \\ v\sin\theta_R \\ w \end{bmatrix} \tag{3.8}$$

Equation 3.8 can also be written in simple form as,

$$\dot{x}(t) = f_c(x(t), u(t)) \tag{3.9}$$

Where f_c denotes the function is represented in continuous time and not discrete time. This method can although be used for only robots/models that have their coordinate frame as the center of the robot and is feasible for the MiR 100 as it satisfies this condition.

The kinematic model of the robot in 3.8 is in continuous time form and needs to be discretized for use in the MPC formulation. This can be done by simple Euler discretization to arrive at the following formulation,

$$\begin{bmatrix} x_R(k+1) \\ y_R(k+1) \\ \theta_R(k+1) \end{bmatrix} = \begin{bmatrix} x_R(k) \\ y_R(k) \\ \theta_R(k) \end{bmatrix} + \Delta t \begin{bmatrix} v(k)\cos\theta_R(k) \\ v(k)\sin\theta_R(k) \\ w(k) \end{bmatrix} \tag{3.10}$$

Where, k is the current step and $k+1$ is the next prediction step. Δt is the sampling period for the data.

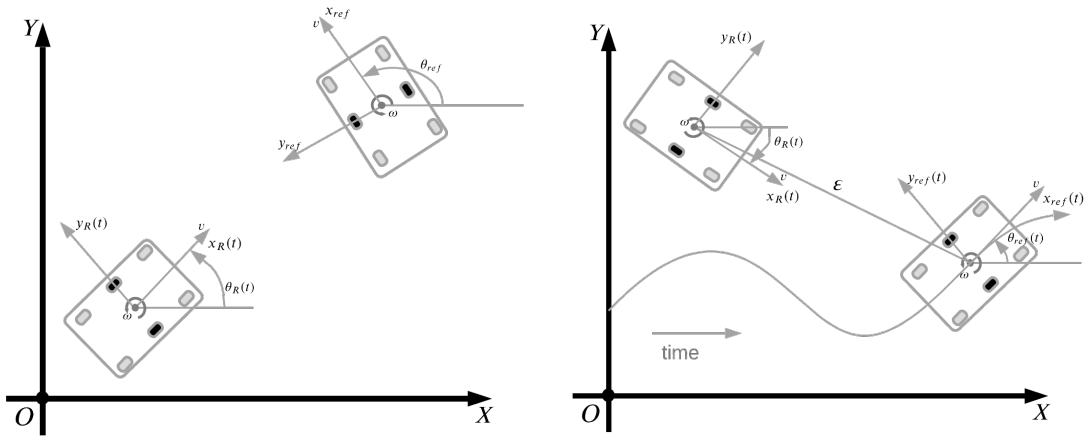
The system model in discrete time can also be written as,

$$x(k+1) = f(x(k), u(k)) \tag{3.11}$$

3.1.2 Model Predictive Control formulation

As described earlier in section 2.4 a model predictive control algorithm is an optimization problem solved at each time step for a given prediction horizon N yielding an optimal control sequence. We have already defined our system model in the previous step. The next step is to proceed with the MPC formulation for the task. An example MPC problem would be similar to the formulation in equation 3.12.

$$\underset{u}{\text{minimize}}\ J(x,u) = \sum_{t}^{t+N} l(x,u) \tag{3.12}$$



(a) ***(b)***

Figure 3.5. *(a) An example point stabilization problem with a constant goal pose* $x_g = \begin{bmatrix} x_{ref} & y_{ref} & \theta_{ref} \end{bmatrix}$*, (b) An example trajectory tracking problem with a goal pose* $x_g(t) = \begin{bmatrix} x_{ref}(t) & y_{ref}(t) & \theta_{ref}(t) \end{bmatrix}$ *changing with respect time* t*.*

subject to a certain set of constraints,

$$u \in U, \tag{3.13}$$

$$x \in X \tag{3.14}$$

For $t = t, t+1, ..., t+N$, N is the prediction horizon, x is the robot state and u is the control variables. $J(x, u)$ is the objective/cost function to be minimized and $l(x, u)$ is the running cost. The above MPC problem is now solved at each time step t for the given prediction horizon N to arrive at an optimal solution progressively.

In this thesis, the robot control problem will be solved as a point stabilization problem. A point stabilization problem is similar to a trajectory tracking problem with the only difference being that for a point stabilization problem, the reference values will remain a constant over the control period[47] as seen in equation 3.15. Where $x_{ref}, y_{ref}, \theta_{ref}$ give the goal pose, denoted by x_g The difference between the two methods can be understood well with the illustration in figure 3.5.

$$x_g = \begin{bmatrix} x_{ref} \\ y_{ref} \\ \theta_{ref} \end{bmatrix}, \forall t \tag{3.15}$$

For an MPC problem it is crucial to define a fitting cost function for the problem to be

solved. As seen in section 2.4, there can be many different methods and approaches to the same problem. These approaches may even make use of deep-learning or Q-learning methodologies as explained in [10, 32, 33]. For a point stabilization problem the cost function can be defined in a very simple manner with the main objective being to minimize the error between the current state of the robot x and the desired goal pose x_g while minimising the control variables u. The objective function for the point stabilization problem can be seen below in 3.16.

$$\underset{u}{\text{minimize}}\ J(x_0, u) = \sum_{k=0}^{N-1} l(x, u) \tag{3.16}$$

Subject to,

$$x(k+1) = f(x(k), u(k)), \tag{3.17}$$

$$u(k) \in U, \quad for\ k \in [0, N-1], \tag{3.18}$$

$$x(k) \in X, \quad for\ k \in [0, N] \tag{3.19}$$

$$v(k) - v(k+1) \leq v_{limit} \quad for\ k \in [0, N-1] \tag{3.20}$$

$$v(k) - v(k+1) \geq -v_{limit} \quad for\ k \in [0, N-1] \tag{3.21}$$

$$w(k) - w(k+1) \leq w_{limit} \quad for\ k \in [0, N-1] \tag{3.22}$$

$$w(k) - w(k+1) \geq -w_{limit} \quad for\ k \in [0, N-1] \tag{3.23}$$

Where,

$$l(x, u) = (x(k) - x_g)Q(x(k) - x_g)^T + u(k)Ru(k)^T \tag{3.24}$$

The equation 3.24 is the running cost for the optimization problem, which is modelled as a quadratic function which includes the robot states and control variables. Q and R are diagonal positive definite weighting matrices of 3X3 and 2X2 dimensions respectively. These matrices are used for tuning the performance of the MPC algorithm. The optimization variables also known as the decision variables for this problem are the state variables of the system x and the control variables of the system u. It is also to be noted that we have only taken into consideration the system kinematics and not the dynamics. This

would mean there are no constraints on the acceleration or deceleration of the robot as mass of the robot is not accounted for. For this reason, we introduce constraints on the maximum allowed difference between the current control inputs $u(k)$ and the control inputs for the next step $u(k+1)$. These constraints can be seen in equations 3.20 through 3.23.

The above equations [3.16-3.23] form the optimal control problem (OCP) for our solution. To be able to solve this problem as an MPC problem, the OCP must be converted into a non-linear programming(NLP) problem. This can be done using various methods, a few of which are listed below,

1. Single Shooting
2. Multiple Shooting
3. Collocation

In this thesis we will be using multiple shooting to convert our OCP problem into an NLP problem. The NLP problem will be solved using Matlab and CasADi[6] through a multiple shooting approach having the decision variables as both the states of the system x and the control variables of the system u.

The MPC algorithm for the formulated problem can be seen in Algorithm 3.1 and the code used for testing the MPC formulation can be seen in Appendix B.1. A simulation run made with sample time Δt as a constant "0.4 s", a selected Q and R value and a fixed x_g value that can be seen in table 3.1 along with the other constant parameters used. The values for the weighting matrices Q and R were selected based on trial and error, with the simulation to obtain the least steady state error for the controller. The steady state error is calculated as the error in distance between the goal x_g and robots final pose. Figure 3.6 shows the path followed by the robot and figure 3.7 shows the control inputs for the solution which resulted in a low steady state error from the simulation test.

3.1.3 Modelling Obstacles as Constraints

The next step is to add the obstacles into the MPC problem formulation. The obstacles in the environment are static poles of diameter $0.45m$. As mentioned earlier, the robot will be maneuvering in a fully known environment. This means the obstacle positions are known and modelled into the MPC formulation with their fixed positions. The pose of the obstacles are denoted as follows,

$$p_{obs}(i) = \begin{bmatrix} x_{obs}(i) \\ y_{obs}(i) \end{bmatrix}, \ for\ i \in [1, N_O] \tag{3.25}$$

Where, N_O is this total number of obstacles.

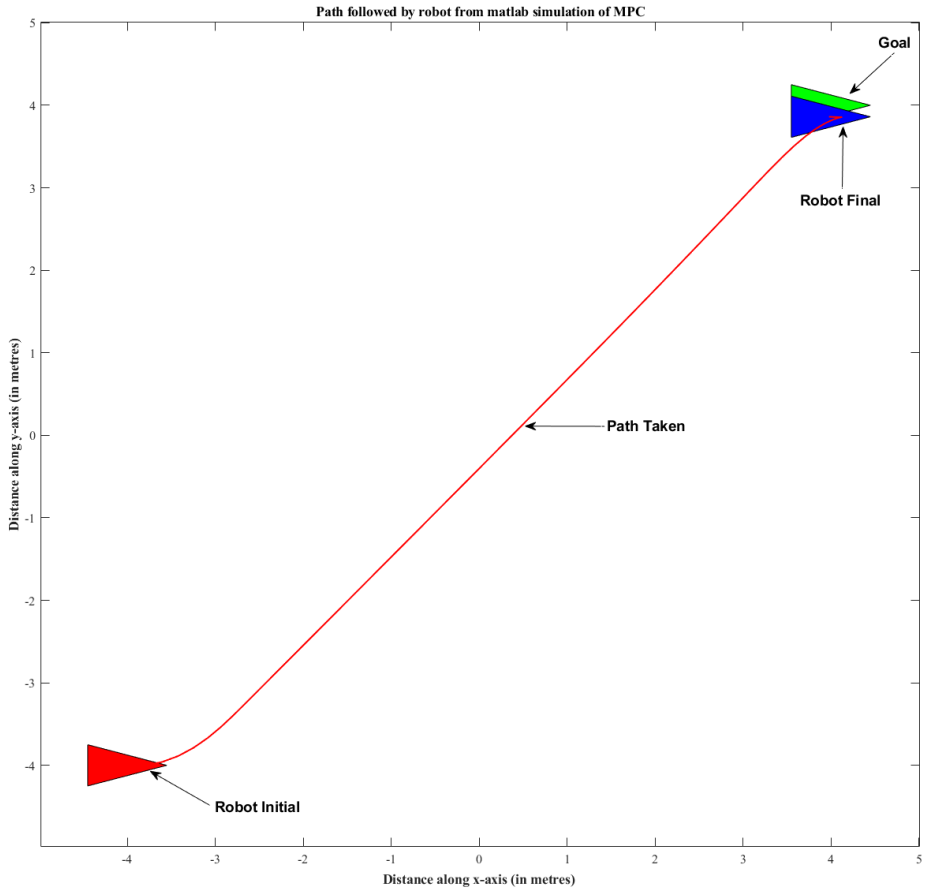


Figure 3.6. *Visualization of the Robot path followed using the MPC algorithm in an environment without obstacle.*

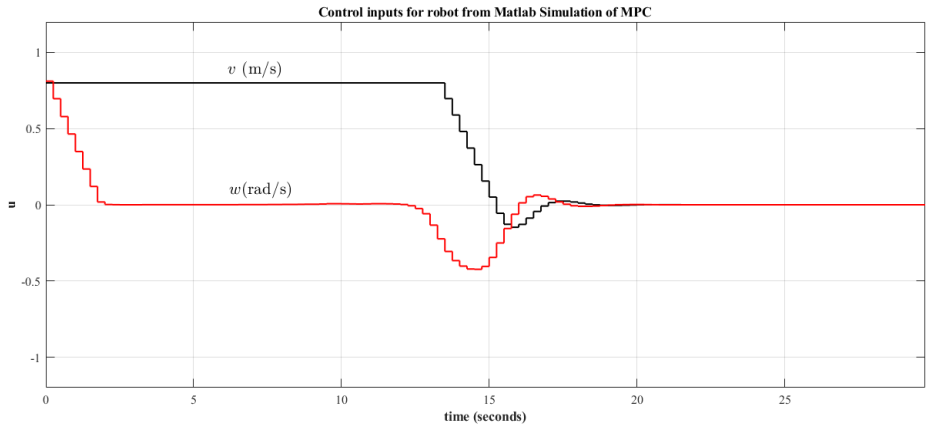


Figure 3.7. *Plot of the control values generated by the MPC for the path in the figure 3.6*

```
Inputs: x_initial[1x3], u_initial[1x2], x_g[1x3]
Parameters: Q[3x3], R[2x2], N[scalar], dt[scalar]
constraints: Boundary constraints, control constraints, ...
             state constraints, control decomposition constraints
Output: U*

% Setting up the MPC problem
define system model as CasADI symbolic
formulate the objective function - obj
define Q[diag] & R[diag]
define optimization variables - x0 & u0
calculate constraints as CasADi symbolic
create the solver using CasADi class 'nlpsol'
input constraint values
input initial value - x_initial
input goal value - x_goal
input initial value for optimization variables - x0 & u0

% Start the control loop
While (x_current-x_goal)>=1e-2 DO
    solve for optimal solution U*
    apply solution of N=1 to the robot
    shift u0,x0,x_curr,t0

end
```

Algorithm 3.1. *Model Predictive Algorithm.*

Symbol and Description	Value
N	15
Δt	0.4
Q [Diagonal elements of Q matrix]	[3 3 1.5]
R [Diagonal elements of R matrix]	[0.7 0.3]
x_g $[x, y, \theta]$	[4,4,0]
$x_{initial}[x, y, \theta]$	[-4,-4,0]
Control lower limits $[v, w]$	[-1,-1] m
Control upper limits $[v, w]$	[1,1] m
Boundary lower limits $[x, y]$	[-5,-5] m
Boundary upper limits $[x, y]$	[5,5] m
Limits on change in linear velocity - v_{acc}, v_{dec}	0.1, 0.1
Limits on change in angular velocity - w_{acc}, w_{dec}	0.1, 0.1
Steady state error (x_g-robot final x)	0.1283 m

Table 3.1. *Parameters and error of the controller in an environment without obstacles.*

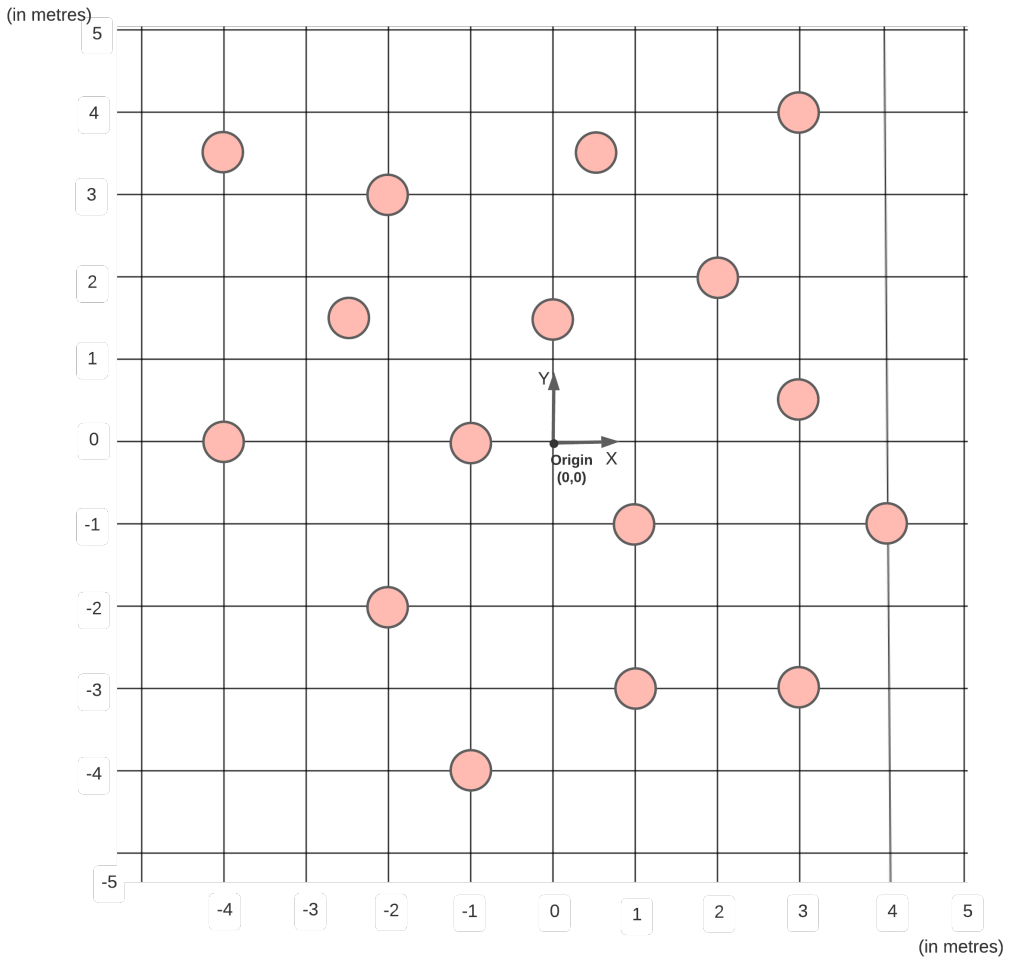


Figure 3.8. *Representation of the static obstacles in the simulation environment.*

The figure 3.8 shows the position of the obstacles in the environment. For the purpose of simplicity, the robot and obstacles are both considered as circles of certain diameter. The robot is modelled as a circle of diameter $0.9m$ since the longest dimension of the robot is $0.89m$. The obstacles are modelled with a diameter of $0.4m$. For the obstacle avoidance to hold true, the Euclidean distance between the robot position [x_R,y_R] and the position of each obstacle p_{obs} must be always greater than the sum of radius of the robot r_R and the radius of the obstacle r_O. An illustration of this can be seen in the figure 3.9 as seen in the equation 3.26.

$$d_{RO}(i) \geq r_R + r_O(i),\ fori \in [1, N_O] \tag{3.26}$$

Where,

$$d_{RO}(i) = \sqrt{(x_R - x_{obs}(i))^2 + (y_R - y_{obs}(i))^2},\ fori \in [1, N_O] \tag{3.27}$$

Here, d_{RO} denotes the Euclidean distance between the robot and the obstacle. This

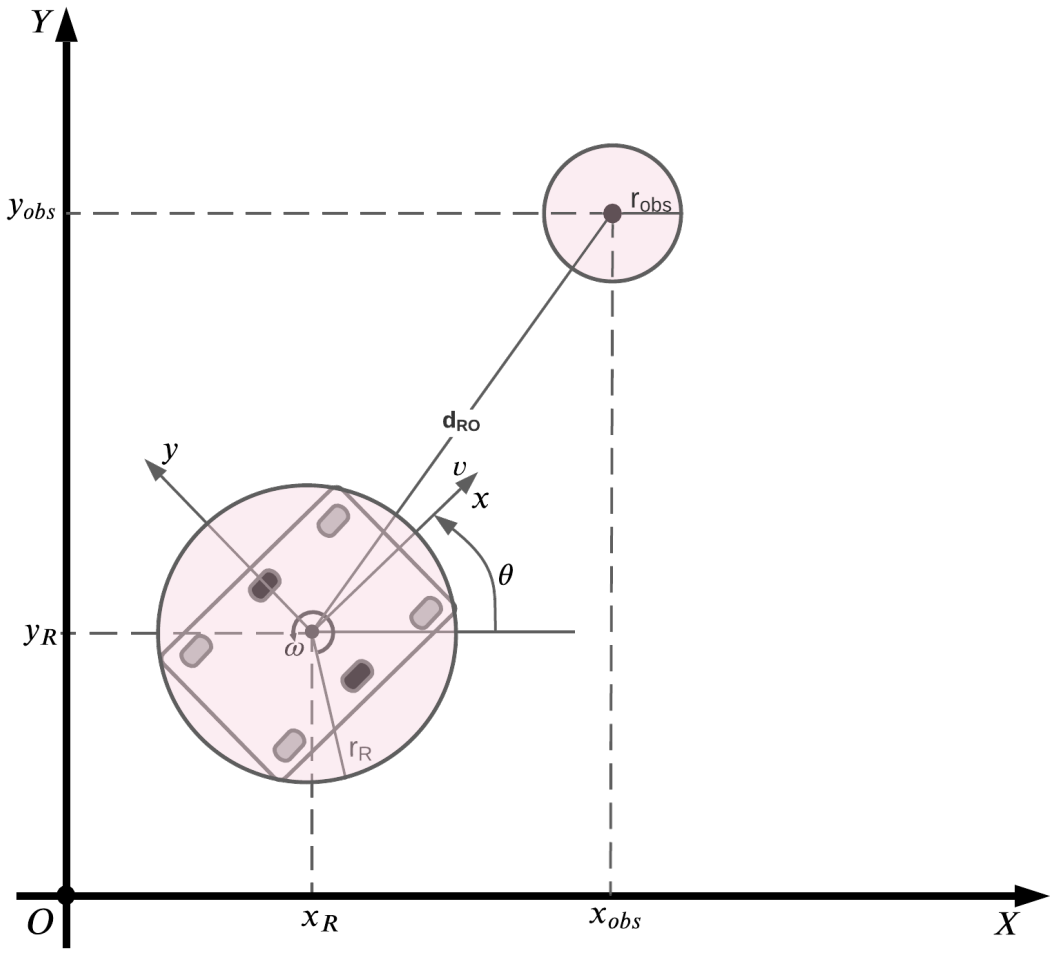


Figure 3.9. *An example representation of obstacle avoidance modelling with a single obstacle.*

condition is now added as a constraint to the MPC problem of the form,

$$-d_{RO}(i) + (r_R + r_O(i)) \leq 0, \quad for\ i \in [1, N_O] \tag{3.28}$$

Where, N_O is the total number of obstacles.

This yields the following final MPC problem formulation,

$$minimize J(x,u) = \sum_{k=0}^{N-1} l(x(k), u(k)) \tag{3.29}$$

Subject to, Subject to,

$$x(k+1) = f(x(k), u(k)), \tag{3.30}$$

Symbol and Description	Value
N	15
Δt	0.4
Q [Diagonal elements of Q matrix]	[6 6 4]
R [Diagonal elements of R matrix]	[0.3 0.15]
x_g $[x, y, \theta]$	[4,4,0]
$x_{initial}[x, y, \theta]$	[-4,-4,0]
r_R	0.425 m
r_O	0.2 m
Control lower limits $[v, w]$	[-1,-1] m
Control upper limits $[v, w]$	[1,1] m
Boundary lower limits $[x, y]$	[-5,-5] m
Boundary upper limits $[x, y]$	[5,5] m
Limits on change in linear velocity - v_{acc}, v_{dec}	0.1, 0.1
Limits on change in angular velocity - w_{acc}, w_{dec}	0.1, 0.1
Steady state error (x_g-robot final x)	0.1142 m

Table 3.2. *Parameters and error of the controller with obstacles included.*

$$u(k) \in U, \quad for\ k \in [0, N-1], \tag{3.31}$$

$$x(k) \in X, \quad for\ k \in [0, N] \tag{3.32}$$

$$v(k) - v(k+1) \leq v_{limit} \quad for\ k \in [0, N-1] \tag{3.33}$$

$$v(k) - v(k+1) \geq -v_{limit} \quad for\ k \in [0, N-1] \tag{3.34}$$

$$w(k) - w(k+1) \leq w_{limit} \quad for\ k \in [0, N-1] \tag{3.35}$$

$$w(k) - w(k+1) \geq -w_{limit} \quad for\ k \in [0, N-1] \tag{3.36}$$

$$(-d_{RO}(i) + (r_R + r_O(i))(k) \leq 0, \quad for\ i \in [1, N_O]\ and\ k \in [0, N-1] \tag{3.37}$$

Phase 2 of the MPC formulation is to to model the obstacles to the controller and test the

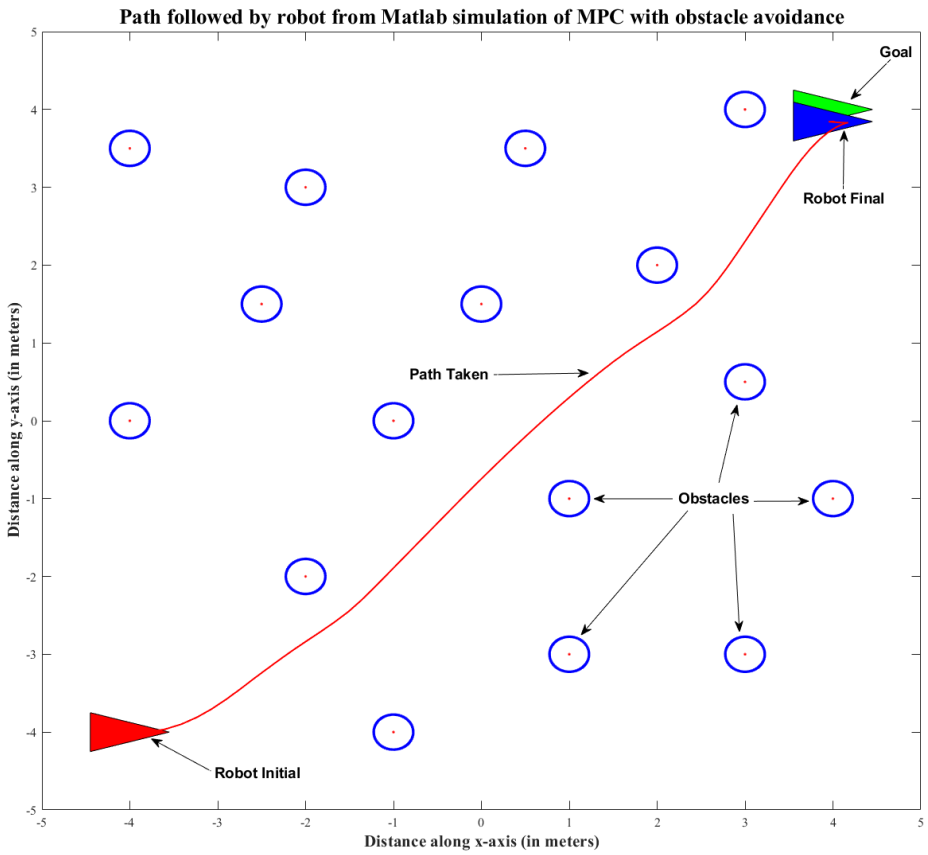


Figure 3.10. *Visualization of the Robot path followed using the MPC algorithm in an environment without obstacle.*

robot control in the environment with obstacles added. The obstacle constraint as seen in equation 3.37 is modelled using CasADi as inequality constraints to the previously defined MPC problem in section 3.1.2. The control algorithm will remain the same as defined in the algorithm 3.1 with only the number of constraints the problem uses will be different. The codes for the obstacle avoidance MPC control scheme can be seen in Appendix B.2.

A simulation run made with sample time Δt as a constant "0.4 s", a selected Q and R value and a fixed x_g value that can be seen in table 3.2 along with the other constant parameters used. The Q and R values were again selected from trial and error method to get a low steady state error, while also avoiding obstacles. Figure 3.10 shows the path followed by the robot and figure 3.11 shows the control inputs for the solution which resulted in a low steady state error from the simulation test.

3.1.4 Inclusion of Human Inputs into the system

The final phase of the MPC formulation for the shared control approach is to complete the shared control strategy by including human inputs to the control loop. In the implementa-

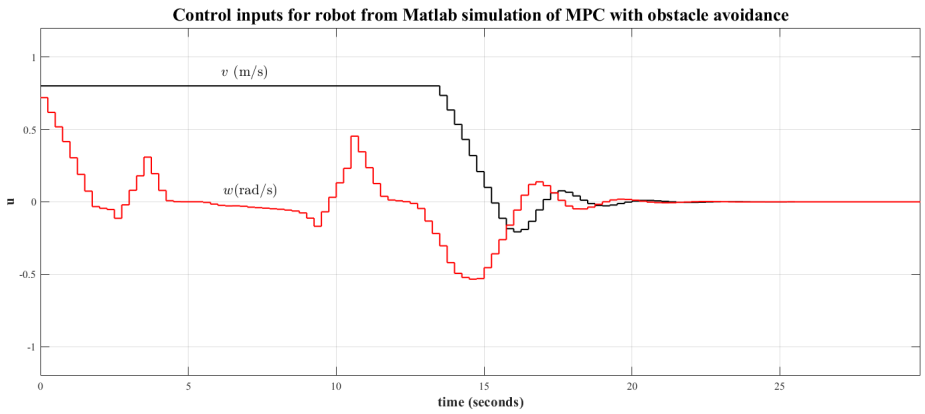


Figure 3.11. *Plot of the control values generated by the MPC for the path in the figure 3.6*

tions discussed in sections 3.1.2 and 3.1.3, the simulation on Matlab were done having a fixed goal/target for the controller to compute an optimal solution.

The shared control methodology developed in this thesis will use human inputs to continuously update the goal pose x_g for the robot. The human therein controls a virtual robot in the environment that enables him/her to effectively manipulate the robot position by changing the target for the MPC controller to solve. The virtual MiR x_{vMiR} controlled by the human has both position $\begin{bmatrix} x_{vR} & y_{vR} \end{bmatrix}$ and direction θ_{vR}, meaning it is represnted as virtual differential drive robot.

$$x_{vMiR} = \begin{bmatrix} x_{vR} \\ y_{vR} \\ \theta_{vR} \end{bmatrix} \tag{3.38}$$

But the human does not have direct control over the pose of the robot but rather controls the virtual MiR using velocity inputs fed to the system as v_h and w_h. The updated pose x_{vMiR} is then calculated using the equation 3.10.

$$x_{vMiR}(k+1) = \begin{bmatrix} x_{vR}(k) \\ y_{vR}(k) \\ \theta_{vR}(k) \end{bmatrix} + \Delta t \begin{bmatrix} v_h \cos\theta_{vR}(k) \\ v_h \sin\theta_{vR}(k) \\ w_h \end{bmatrix} \tag{3.39}$$

To include human inputs we use the discrete time model of the kinematic system defined in equation 3.10 to update the new desired goal position of the robot. To better formulate the control structure for inclusion of human control, the below 2 assumptions are made,

1. The virtual MiR controlled by the human is restricted to move within a fixed distance d_{PR} from the real robot which is illustrated in figure 3.12

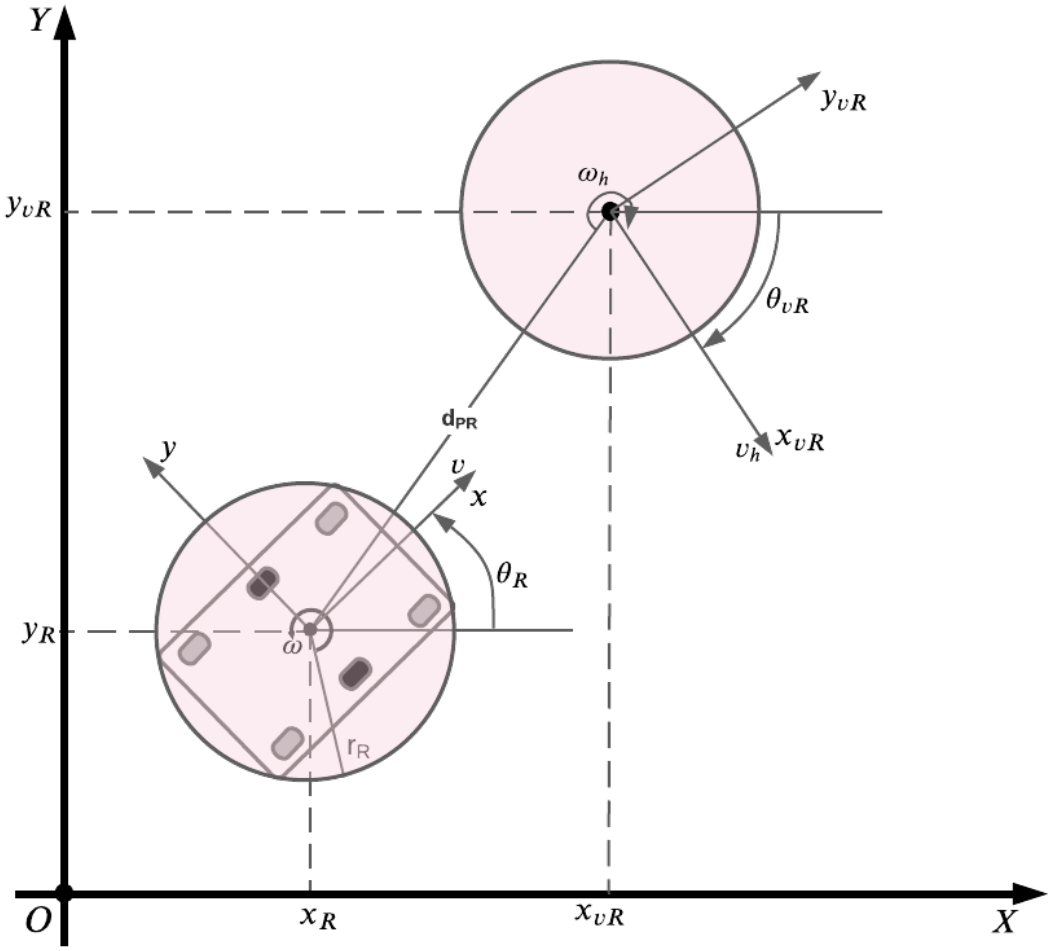


Figure 3.12. *A representation of the control point controlled by the human at a distance* d_{PR} *in front of the robot.*

2. The virtual MiR is allowed to pass through the obstacles and outside the specified environment, while the real robot cannot.

The cost function as defined in equation 3.24 can be seen to be working on a fixed/pre-defined goal pose x_g. This cost function can now be slightly modified to include to the human controlled virtual MiR for the proposed shared control strategy.

$$l(x,u) = (x(k) - x_{vMiR})Q(x(k) - x_{vMiR})^T + u(k)Ru(k)^T \tag{3.40}$$

3.2 Shared Control Implementation

3.2.1 The Simulation Environment

In this chapter, so far, we have defined the kinematic model of the robot used for the thesis, which is the MiR100[27]. We have modelled the MPC problem without obstacles and tested it on Matlab simulation. We have also later added obstacles and modelled them into the MPC formulation as constraints. Finally, we defined and added a model for including human into the control loop. The next step is to test the MPC formulation on the MiR100 robot.

To perform the simulation, different simulation environments were tested, which include:

1. Microsoft AirSim using Unreal Engine [37]
2. Gazebo Simulation [19]
3. Isaac SDK and Isaac sim from Nvidia

The two most appropriate simulation environments out of the list were the Microsoft AirSim and Gazebo simulation environment. Initial tests began working with AirSim simulation environment. AirSim simulation environment was developed for easy testing and deployment of Reinforcement Learning(RL) based algorithms for autonomous vehicles and drones. A major drawback with the AirSim simulation environment is that, it works only through Microsoft Windows Operating Systems.

A simple pipeline of the AirSim environment and it's different modules can be seen in the figure 3.13. The simulation module seen on the right side of the image can be compiled with the AirSim library package into an executable "`.exe`" file. AirSim provides an API for communicating with this simulation environment through both python and C++ programming languages. AirSim API provides ready classes for implementation of algorithms for control of Drones and vehicles that use a bicycle model.

In our thesis the main focus is working towards an implementation for differential drive mobile robots. This would mean, simulating a robot with skid steering or differential drive would require developing a code from scratch, for control of differential drive vehicles. Due to this main reason, use of the AirSim simulation environment was dropped and simulation were planned and executed using Gazebo simulation environment

Gazebo is an open source, high fidelity 3D multi-robot simulation environment that comes pre-built with a full desktop installation of Robot Operating System (ROS)[35]. Gazebo provides capabilities for data visualization by simulation of remote environments and hardware. Gazebo simulation environment is easy to use with possibility of environment creation via a Graphical User Interface (GUI). All models used in the simulation environment are developed using Unified Robot Description Format (URDF) and Simulation Descrip-

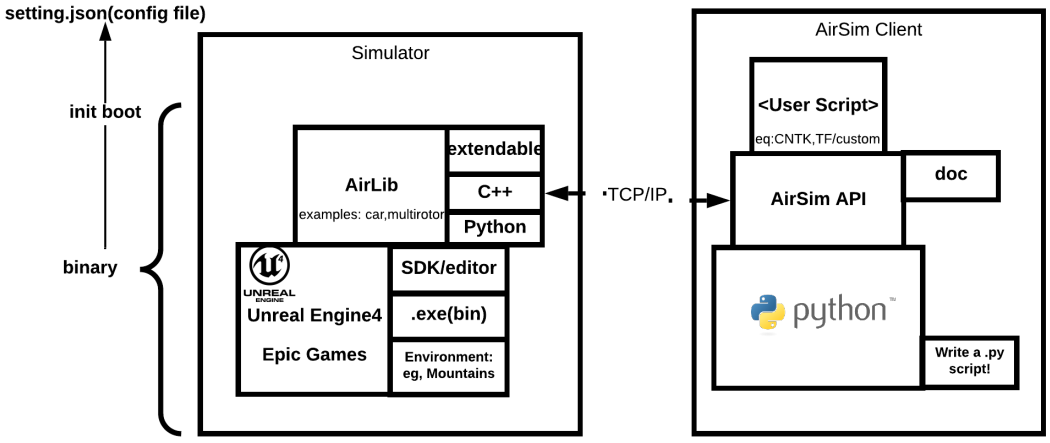


Figure 3.13. *Pipeline illustrating AirSim's modules and communication between them.*

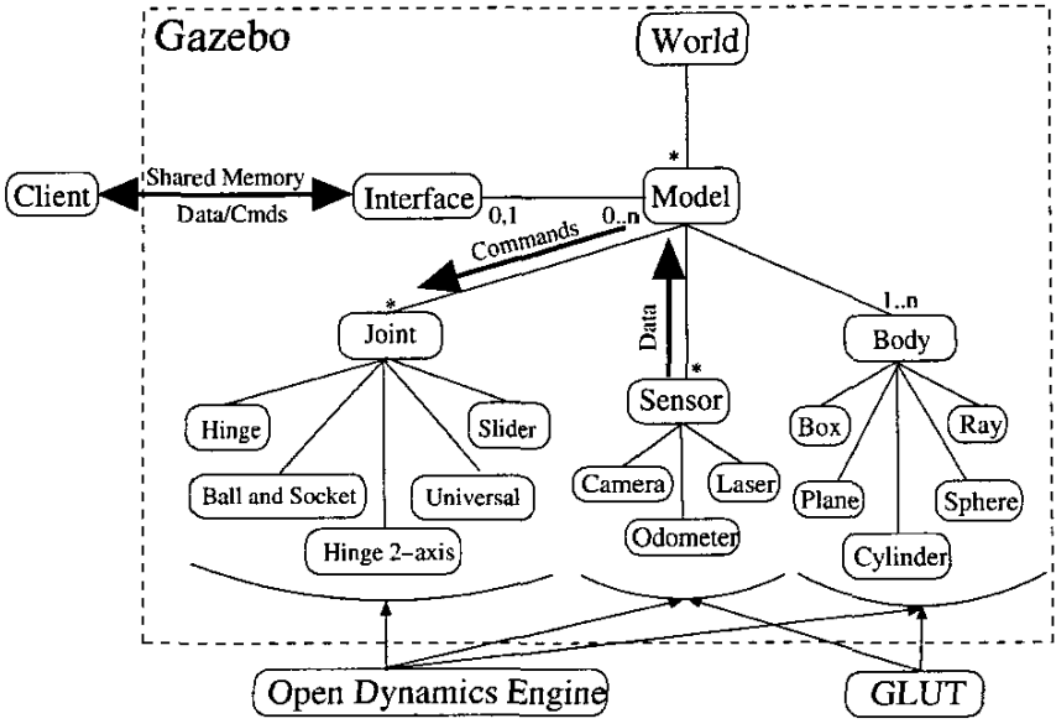


Figure 3.14. *General structure of the Gazebo simulation environment components [19].*

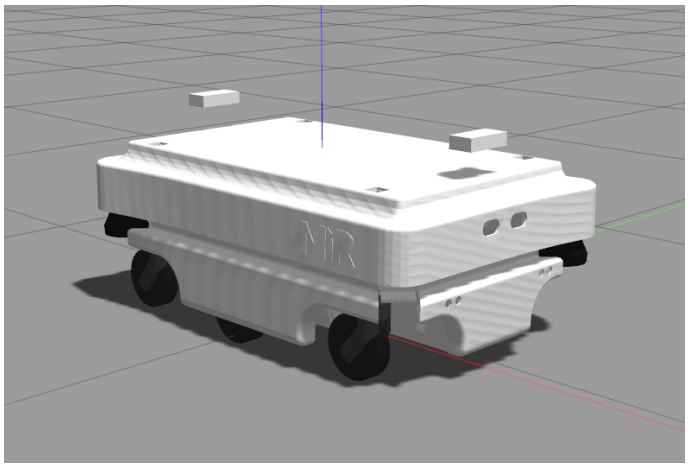


Figure 3.15. *The modified MiR robot with cameras.*

tion Format (SDF). Both these formats use an XML script base for model creation. A general structure of Gazebo can be seen in figure 3.14. For implementation, in this thesis, we will be simulating the MiR100 mobile robot using the Gazebo simulation environment. We will not be developing the model, but rather, we wil be using an existing model developed by DFKI, the German Research Center for Artificial Intelligence[8].

The MiR model was modified to include two cameras, one camera to visualize what is in front of the robot, and another placed at the rear of the robot to visualize the what is behind the robot. The modified configuration file can be seen in appendix A. The modified robot model can be seen in figure 3.15, where the two rectangular boxes hovering over the robot represent the cameras of the robot. The simulation environment used for the thesis was created using the Gazebo GUI. The designed simulation environment was such that it mimics a robot moving in an environment with many trees. This will help in testing the maneuverability of robot using the implemented shared control approach. While designing the environment it is assumed that the robot will be moving in a completely flat environment with no humps or crevices. The designed simulation environment can be seen in figure 3.16.

Equation 3.8 gives a representation of the standard kinematic model for most differential drive mobile robot. The localisation of the robot can be done using odometry data from the wheel encoders or using other methods like visual odometry using LIDAR sensors or cameras. But in this thesis we get the position of the robot with respect to the world coordinate through the Gazebo simulation environment directly. This is done by subscribing to the Gazebo's /Get_model_states topic.

3.2.2 Completing the Pipeline for Shared Control Implementation

This chapter has so far walked through all the different parts that are required for implementing the shared control approach of this thesis. With the MPC problem formulation

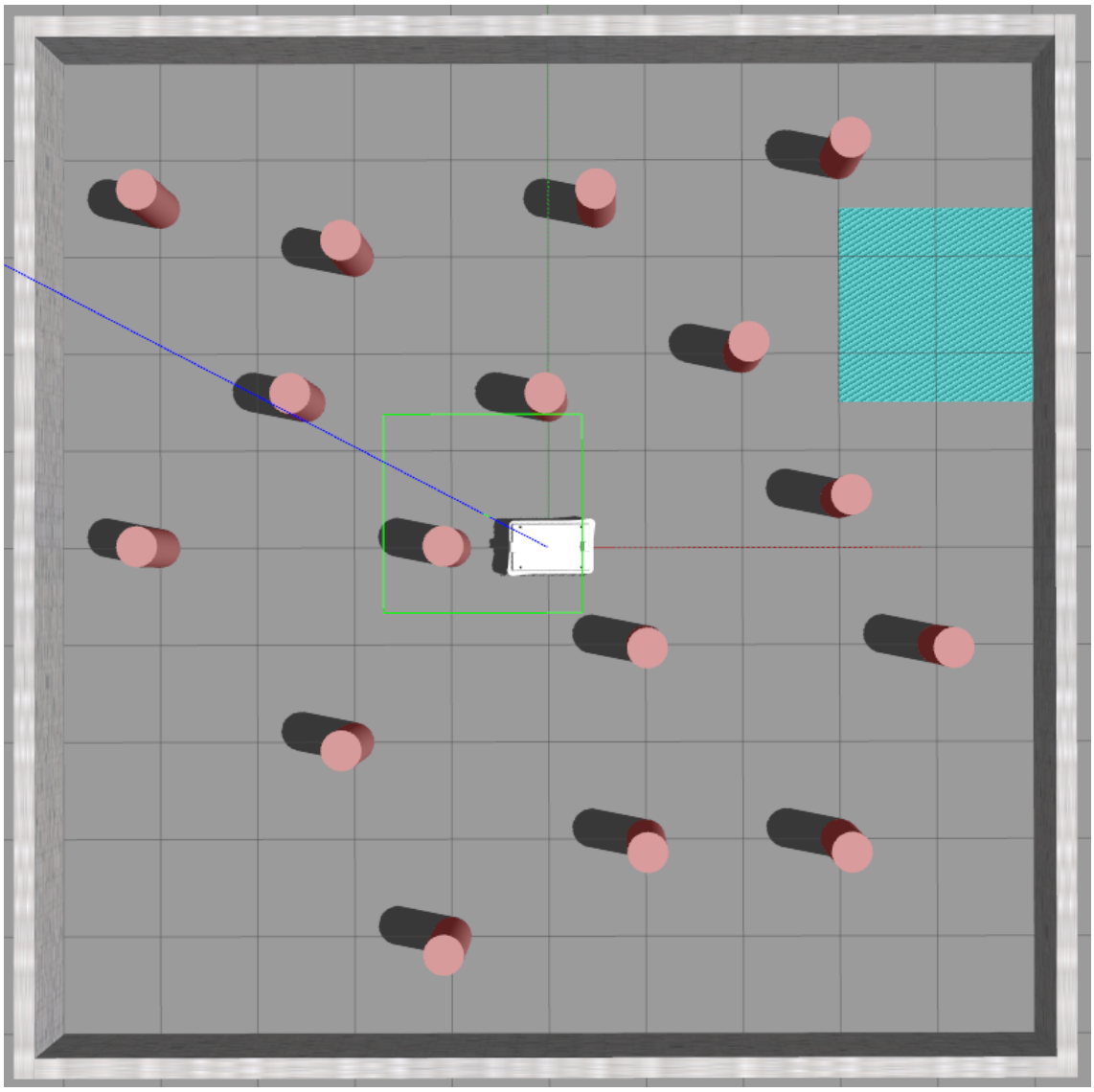


Figure 3.16. *The Gazebo simulation environment created for testing the shared control approach.*

and the simulation environment ready, the next step is to put things together to complete and test the implementation of the shared control algorithm. As seen in the figure 3.1, all the different components implemented will communicate with each other using ROS. ROS is an open source meta operating system that works on top of Linux and Mac Operating Systems. It can be seen as a set of tools, libraries and conventions put together to simplify robotics applications and enable rapid prototyping and testing of robotic software[35]. ROS provides APIs for multiple programming languages including python, C++ and Matlab giving complete freedom to the user to choose the preferred coding language. ROS also has its own messaging medium using publish subscribe services which let's codes written in different languages to still communicate with each other.

In this thesis, the implementation of the shared control algorithm is done using Matlab and python. As mentioned in section 3.1.2, the MPC algorithm is written on Matlab, using

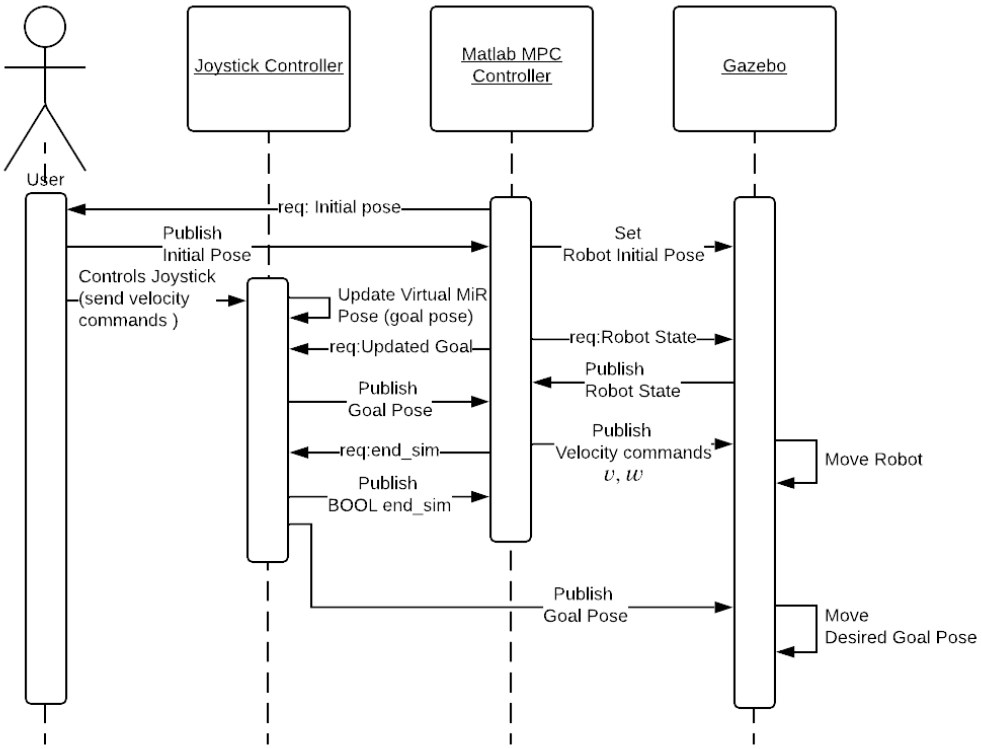


Figure 3.17. *Sequence Diagram of the Different modules and their communications.*

the CasADi optimization toolbox. The human inputs are collected using a joystick through python and the data is published onto ROS. The joystick in use is the Logitech ATTACK 3 joystick[23]. This is a joystick with 2 DOF where the Y axis of the joystick controls the linear velocity v and the X-axis of the joystick controls the angular velocity w.

The user will use joystick with the model defined in section 3.1.4 to provide the control inputs, which will move the goal to reach of the robot within the simulation environment. The pose of the desired goal positions is published as ROS messages. These ROS messages are subscribed by Matlab and the goal to reach is continuously updated for the MPC algorithm. Matlab also subscribes to the current pose of the robot in the simulation environment and updates the value into the MPC algorithm. Matlab uses these values to then calculate a solution for the defined MPC problem. The solution of the MPC algorithm will include the control inputs for the robot throughout the whole prediction horizon. We will extract only the values for the next step and publish these control values v and w as a ROS message for the robot to subscribe to and move accordingly in the gazebo simulation environment. The algorithm for the proposed shared control approach can bee in algorithm 3.2. An illustration of the different modules and their communication can be seen in the form of a sequence diagram in figure 3.17.

The implemented algorithm was tested using the same parameters defined in the table 3.2. The algorithm was tested with some sample runs and the robots behaviour against

```
1   Inputs: x_initial[1x3],u_initial[1x2],x_g[1x3](form user)
2   Parameters: Q[3x3], R[2x2], N[scalar], dt[scalar]
3   constraints: Boundary constraints, control constraints,...
4                state constraints, control decomposition constraints
5   Output: U*
6
7   % setting up ROS parameters
8   rosinit
9   initialize the ROS publishers and message types
10  initialize ROS subscribers
11
12  % Setting up the MPC problem
13  define system model as CasADI symbolic
14  formulate the objective function - obj
15  define Q[diagonal] & R[diagonal]
16  define optimization variables - x0 & u0
17  calculate constraints as CasADi symbolic
18  create the solver using CasADi class 'nlpsol'
19  input constraint values
20
21  x_initial -> from user
22  x_goal = x_initial
23
24  robot pose in gazebo->Ros Service call:pose->x_initial
25  subscribe to terminate button: end_sim -> Bool
26
27  input initial value for optimization variables - x0 & u0
28
29  % Start the control loop
30  While end_sim==false DO
31      solve for optimal solution U*
32      rospublish the solution of N=1 for the robot
33      shift u0,x0,t0
34      rossubscribe robot pose->x_curr
35      rossubscribe goal_pose->x_goal
36  end
```

Algorithm 3.2. *Shared Control Algorithm*

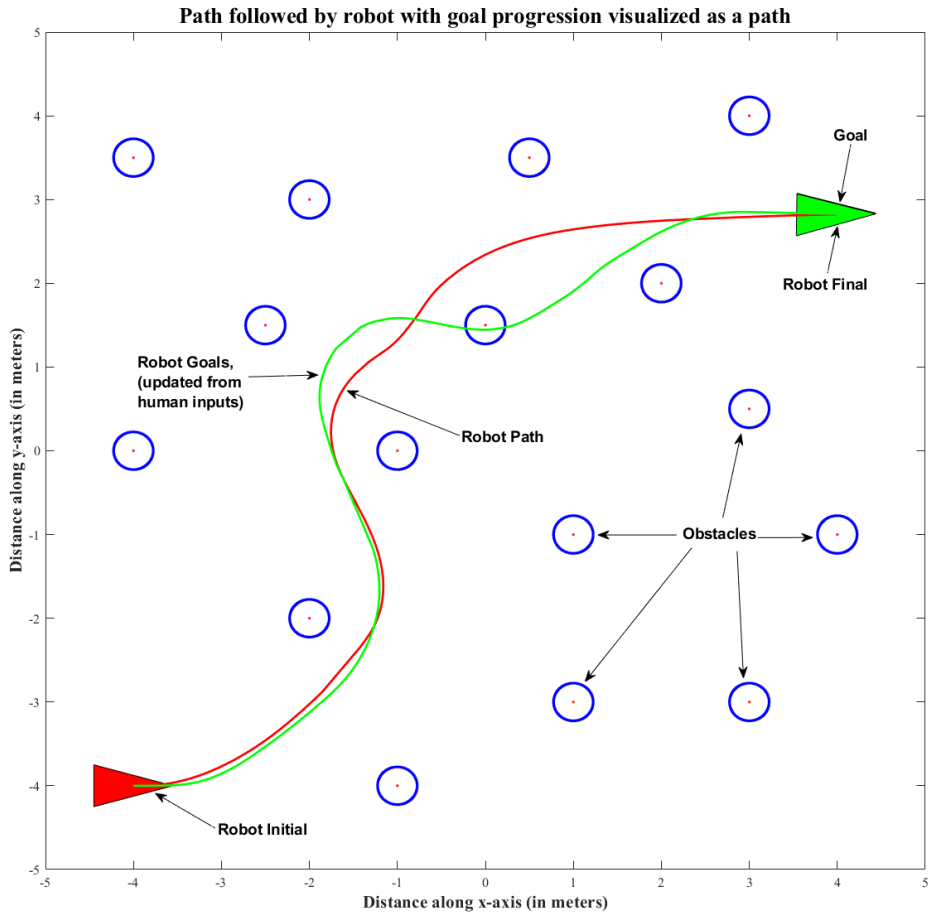


Figure 3.18. *Visualization of the Robot path followed using the Shared Control algorithm in the gazebo environment.*

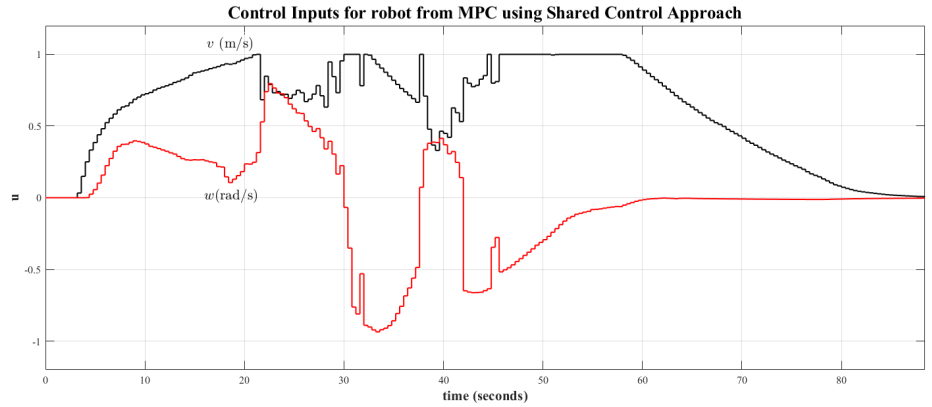


Figure 3.19. *Plot of the control values generated by the MPC for the path in the figure 3.18.*

obstacle avoidance was tested. The path followed by the robot and the control inputs generated by the MPC algorithm from one of the sample test is shown in figure 3.18 and figure 3.19 respectively. The robot does not follow the path chosen by the human but rather tries to move to the frame defined by the virtual robot at a given time as seen in figure 3.18. This is because the MPC formulated solves for a point stabilization problem.

4 USER TESTING AND RESULTS

In the previous chapter the shared control architecture using an MPC approach was formulated and implemented. The implemented architecture now needs to be tested for evaluating it's performance. To arrive at an unbiased evaluation the system is to be tested with users of different experience levels and backgrounds. This would help to come to a proper conclusion of how the implemented shared control architecture behaves and feels in terms of usage.

4.1 Test Procedure

4.1.1 Manual tele-operation

To arrive at a proper performance of the system we need to have a benchmark. For the purpose of having a benchmark test, a manual tele-operation module was designed to be used in the gazebo simulation along with the MiR robot. Manual/Direct tele-operation was chosen due to the reason that many systems that require human operators to perform remote tasks function with direct tele-operation methods using control interfaces that mimic the remote machines control interface.

The manual tele-operation module uses the joystick model adapted for the shared control approach. Now that the user will operate the robot directly, the joystick controller directly sends the required velocity commands for the robot to move within the environment and not the virtual MiR. The joystick control module used can be seen in the Appendix B.6. The code subscribes to the ROS node Joy that publishes the raw joystick data. This data is then converted to a ROS message of type Twist with the linear velocity v and angular velocity w values mapped from the joystick axes. The Twist message is then published as a rospublish message that will be subscribed by the robot in the simulation environment. It can also be noted that a joystick button is required to be pressed at all times while wanting to move the robot. This is used as a safety switch/deadman switch to restrict undesired or accidental movement of the robot.

4.1.2 The user testing tasks

Now that we have a bench marking medium, we will need to proceed with defining suitable tasks to be performed as part of testing the shared control architecture. For the user testing, two tasks were defined.

1. **How safe can you manoeuvre?**
   In this task, the users will manoeuvre the robot randomly to their will around the simulation environment that can be seen in figure 3.16. The users will perform this task for a specified time-frame of 2-minutes. 2-minutes was chosen as that would also give an idea on how the user adapts to the system and the environment.
2. **How quick can you manoeuvre?**
   In the second task, the user will have to manoeuvre from a starting point in the environment, to, a highlighted region on the environment as quick as they can be.

The performance of the shared control approach will be tested and compared against a manual tele-operation methodology based on a set of performance evaluation criteria that can be seen below.

1. **Task 1 Evaluation criteria**
   - The total number of collisions made per minute.
2. **Task 2 Evaluation Criteria**
   - The time taken for moving from starting position to the final highlighted region. This highlighted region can be seen in the figure 3.16 as the blue shaded region on the top right corner of the simulation environment.
   - Does a small training period of using the system, improve efficiency of task completion?

     The task will be performed twice with the implemented shared control approach. The first time users will directly perform the task. This will be followed by a 3 minute training period for the users to get accustomed to the system. After this training period, the users will perform the task again to see if there is an improvement to their performance.

4.2 Test setup

The user tests were set up and performed on the TAU, Hervanta campus. The test environment is setup using gazebo as shown in the figure 3.16. The users were provided with video feeds of the robots front and back facing cameras as depicted in figure 4.1 and 4.2 respectively. For the manual tele-operation task, these were the only medium of perception that was provided.

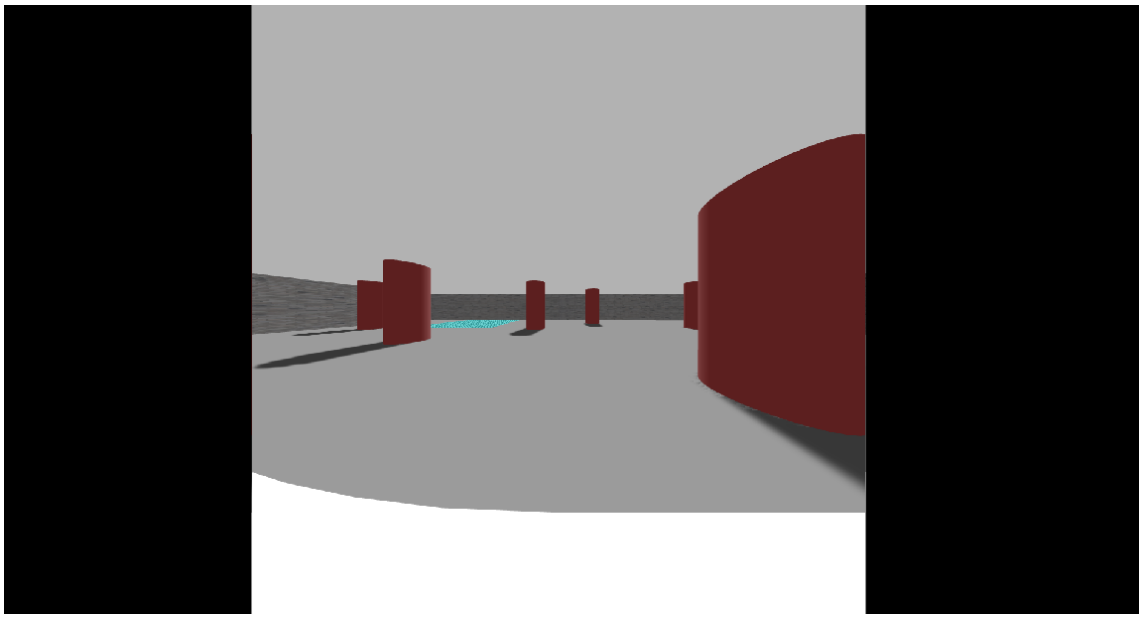


Figure 4.1. *Front camera feed of the robot used as a display for the user testing.*

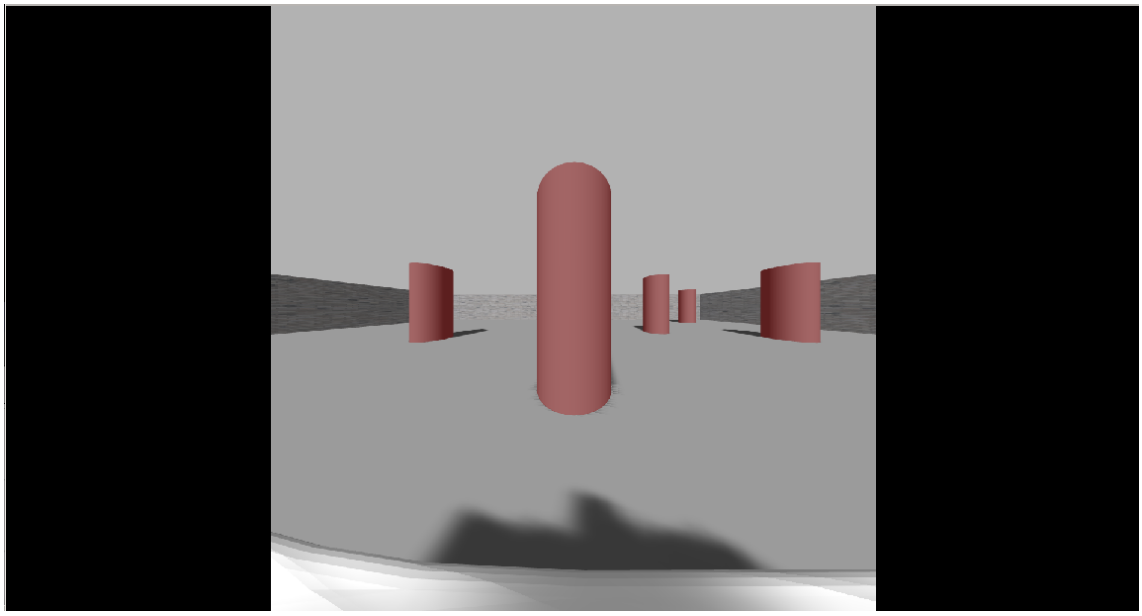
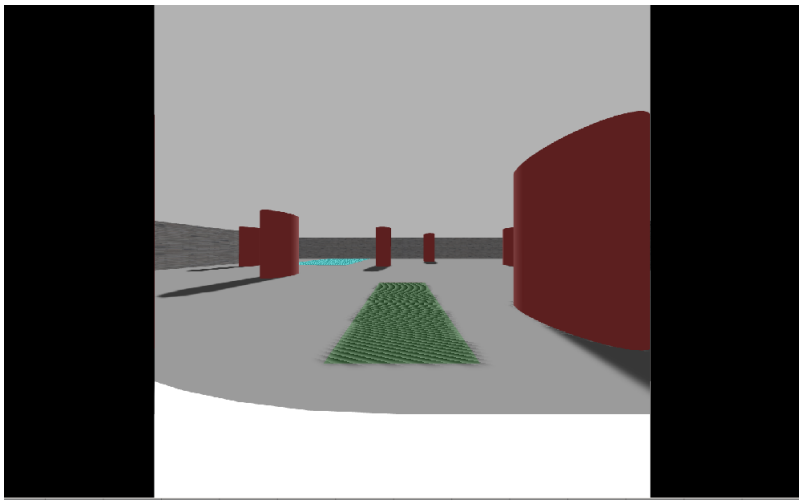
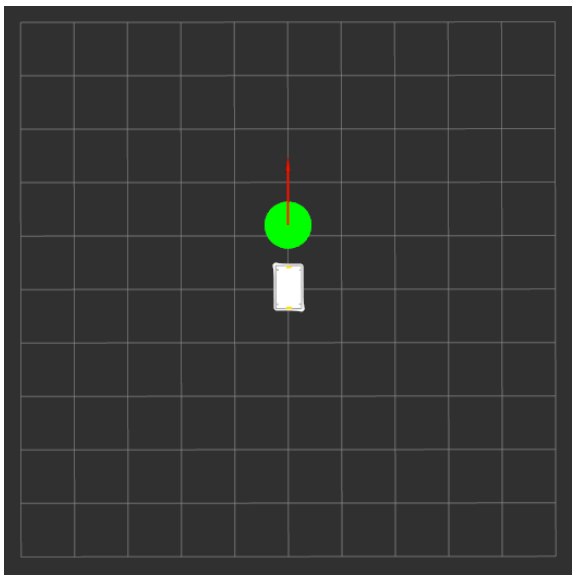


Figure 4.2. *Rear camera feed of the robot used as a display for the user testing.*

For the tele-operation tasks with shared control, the user will be controlling a virtual MiR in the environment that is represented by the green box in the figure 4.3(a). Having just the two camera feeds can prove to be daunting at times when the robot is directly on top of the Green rectangle object that the user controls. The user will not have enough information or knowledge as to how much he has turned the point or will not have a view of the controlling point when he moves it to the sides as there are no visual aids for what is on the side of the robot. To help the users in such situations, the pose of the virtual MiR with respect to the robot's position is visualized on Rviz. Rviz is a powerful data visualization tool that comes pre-installed with ROS. It allows for users to subscribe



(a)



(b)

Figure 4.3. *(a) A view of the user controlled pose on the environment for the implemented Shared Control Approach. (b) A representation of the pose of the user controlled point with respect to the robot provided as an additional visual aid for the users.*

to and visualize a lot of the ROS messages being published along with transformations between different entities[17]. A visualization of the user controlled virtual MiR pose with respect to the robot position can be seen in the figure 4.3(b). An over all view of the complete test setup can be seen in figure 4.4.

4.3 Results

A total of 7 users were selected for the user testing tasks. The group consisted of 2 female participants and 5 male participants. Out of the 7 selected participants 3 had very good gaming experience and hence good experience with knowing how to control the robot. 4 participants were from an automation or robotics background while the remaining 3

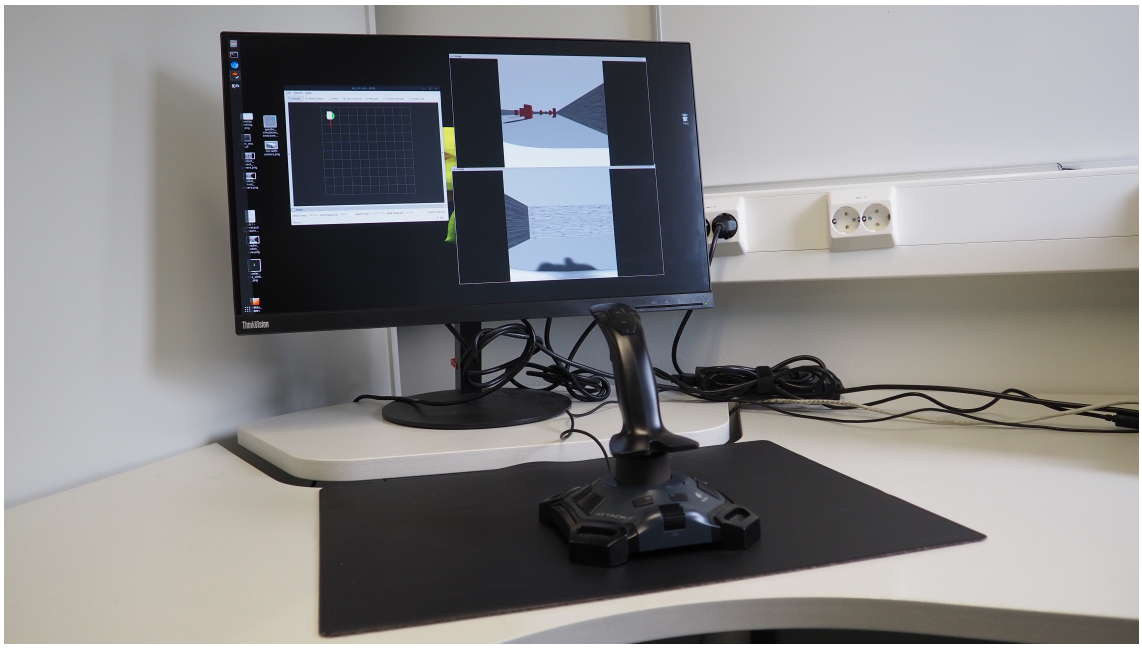


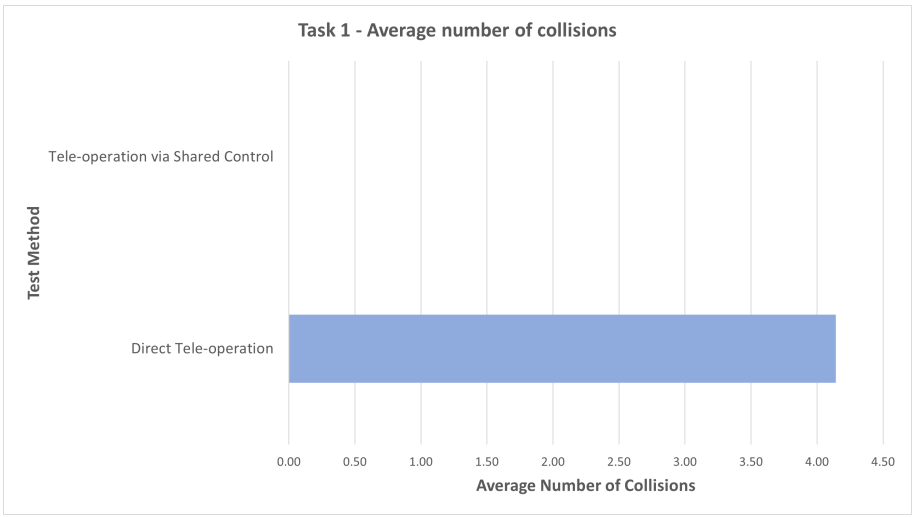
Figure 4.4. *The complete user test setup at Tampere University, Hervanta Campus.*

participants were from other domains. The users were verbally instructed how to use the system and what they will be able to see in the screens in front of them. All user testings made were voluntary and an informed consent form was signed before each user test. The template of the consent form can be seen in appendix C.1.

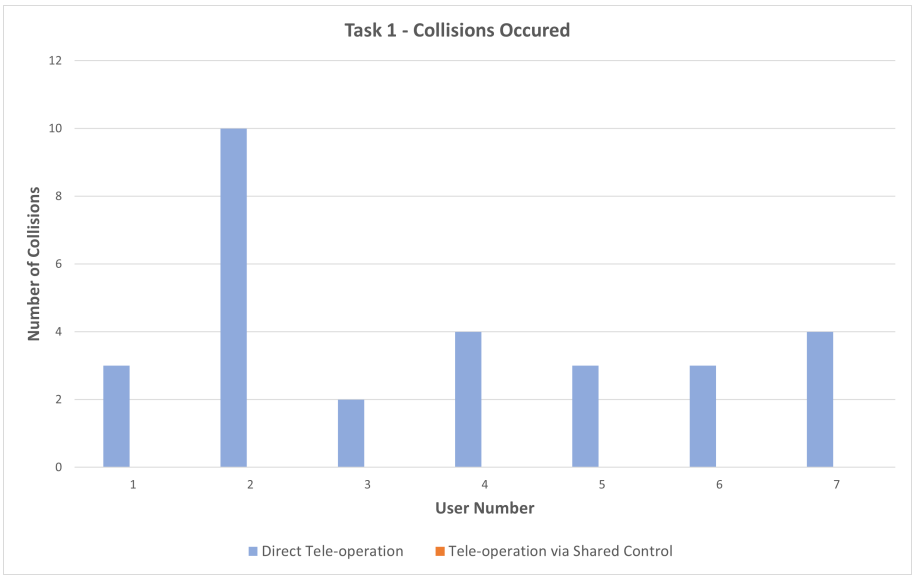
The user test were performed for both tasks and each test was evaluated based on the evaluation criteria mentioned in section 4.1.2. The evaluation provides a comparison of the task completion in terms of maneuverability and safety for the system that is being used. During both the tests the robots maximum allowed velocities were restricted to 1 m/s.

4.3.1 Task 1 - How safe can you manoeuvre?

An average of the results from all 7 of the user studies can be seen in figure 4.5(a). The results are from an average of the collisions made by all the users in the environment over the fixed time period of 2-minutes. Figure 4.5(b) provides the data of collisions per user. From the plots it can be seen that a couple of users, had a lot of trouble maneuvering the robot in the environment with manual tele-operation method. Though 2 of the users performing the tests had good experience using systems with joystick control, all the users ended making at least 2 crashes in the environment. With the shared control approach there was no collisions made by the robot in the environment.



(a)

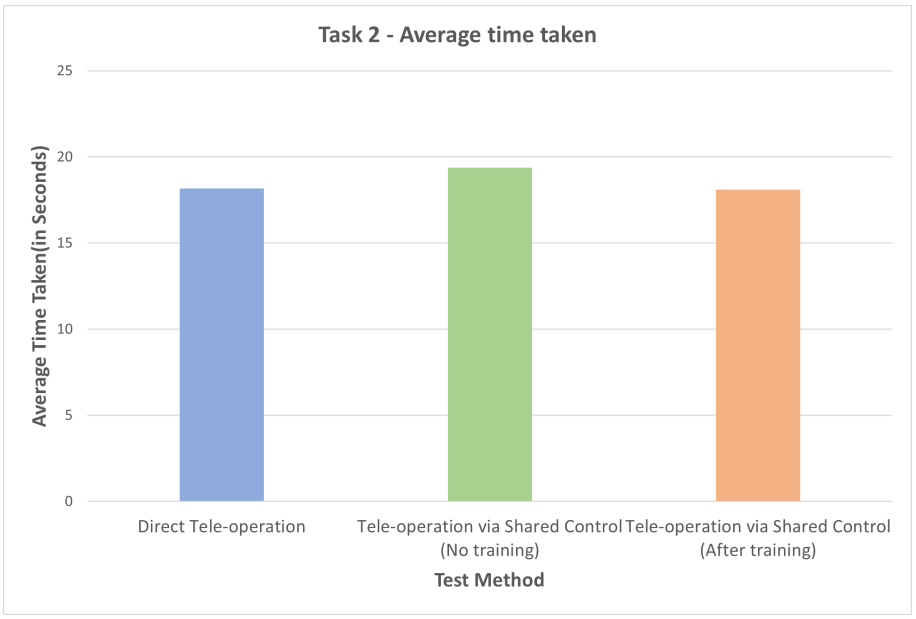


(b)

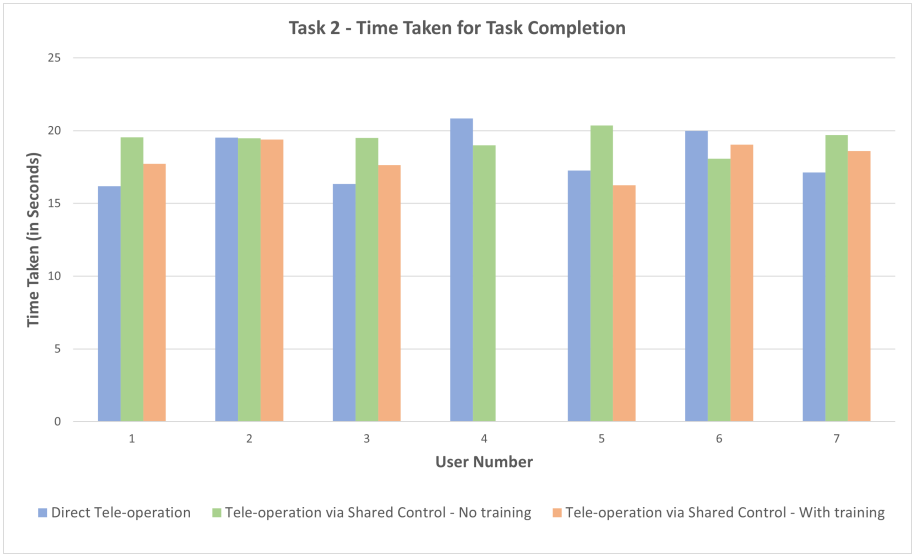
Figure 4.5. Task 1 *(a) A plot depicting the average number of collisions occurred in 2 minutes using manual tele-operation and shared control. (b) Plot showing the number of collisions per user for the 2 minutes that the task was performed.*

4.3.2 Task 2 - How quick can you manoeuvre?

An average of the results from all 7 of the user studies can be seen in figure 4.6(a). Figure 4.6(b) shows the time taken per method for each user performing the test. For this task the main evaluation criteria was to check for the efficiency of performance of using a shared control approach against a manual tele-operation method. This task was done in 2 phases. Initially the users performed the task of moving the robot to a highlighted region in the environment using both the methods of operation. From the plots we can see that the average time taken to complete the task using shared control approach is higher than when performing the task with direct tele-operation.



(a)



(b)

Figure 4.6. Task 2 *(a) A plot depicting the average time taken to complete the given task using manual tele-operation and shared control with and without a training period. (b) Plot showing the time taken per user for task completion using manual tele-operation and shared control with and without a training period.*

The second phase, users had a 3 minute training time to get accustomed to how the system behaves. After this training period the users performed task 2 using the shared control approach again. This time almost all the users had an improved time in comparison to not having any training time for using the system. It can also be seen that, most results post training are close to the results from direct tele-operation. In certain tests, the shared control approach yielded faster task completion times post training.

5 ANALYSIS

A shared control approach as defined in section 2.1 can have various different methods of implementation. The method proposed in this thesis was tested and the results of the implemented shared control approach were presented in the previous chapter. A short discussion with the participants of the user tests revealed that, a lot of them found the shared control approach to be a little bit difficult to grasp in terms of controlling the robot with only the camera view. For users who had no prior experience using a joystick to control machines, this approach felt overwhelming as they felt the need to adapt to a system rather rapidly. This would mean, in terms of reducing the cognitive load from the user this method would not be the most appropriate until the user learns to adapt how to use the system. From the user testing, it was prominent that some users did feel a lot more comfortable to control the robot after they were given time for training and getting accustomed to the systems behaviour.

5.1 Shortcomings

The proposed approach takes longer times of task completion and in a general context the robot control is slower while compared to using a traditional tele-operation control. But how fast and nifty a system can complete tasks is not the major criteria in most cases. The major criteria whilst working with heavy and expensive machinery falls to the major concern of safety of both the machinery and the operator(e.g. Forestry and mining tasks). From the results it can be seen that using a shared control approach largely benefits in reducing human errors, whether they be intentional or by accident.

The best driver and worst driver results from the tests of task 2 can be seen in the figure 5.1. The best and worst driver are considered based on the criteria of how quick the user was able to perform the task using manual tele-operation. The best driver has the least time for manual tele-operation, while the worst has highest time for task completion using manual tele-operation. From tests of both the drivers, it can be seen that the safety aspect of the implemented shared control approach is always achieved as seen in the plot in figure 4.5. With the best driver, a person who knows well to tele-operate and handle a joystick, we can see that the user can operate quicker with a manual tele-operation approach as compared to using the implemented shared control approach. While the

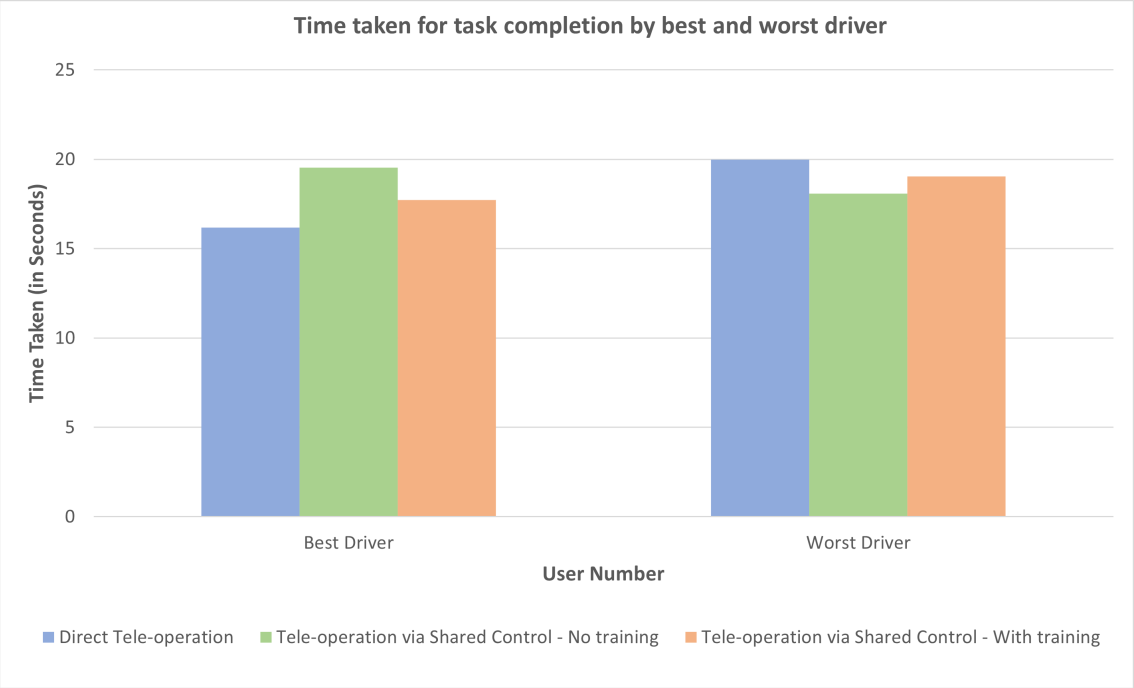


Figure 5.1. *Plot depicting the time taken for task completion by the best driver and worst driver*

latter user performs the task more efficiently using the shared control approach as seen in the plots in figure 5.1.

The MPC formulation

The slower task completion times of the shared control approach can be linked back to the MPC problem formulation used for the implementation of the share control architecture. The objective function used in the thesis defined in equation 3.24 is a very straightforward quadratic formulation of a point stabilization problem. This does not take into consideration any lateral slippage of the robot in real time nor does it currently include any terminal costs to the problem formulation.

Inclusion of these into the objective function will increase the problem and help in a much better control of the robot. Another approach would be to transform the point stabilization problem into a trajectory tracking or path following problem, in which the user movement creates a path for the robot to follow. Sampling time of the MPC control loop and the values of the diagonal weighting matrices also play a crucial role in the performance of the controller. These values are currently determined using trial and error in this thesis. Ideally having a mathematical or machine learning approach to finding the best suited values for the weighting matrices will help in improving the performance of the MPC controller.

Obstacles avoidance formulation

Having a proper obstacle avoidance control is crucial for the safe operation of the machinery. Although the obstacle avoidance implemented in the proposed shared control approach works for most times, but, it is not flawless. During user tests it was observed that the robot never hits an obstacle when the user moves the goal position constantly, but eventually sometimes the robot gets stuck and cannot find it's way around certain obstacles due to the way the system is modelled. The robot is currently modelled as a circle for the obstacle avoidance, while the actual robot is if a rectangular shape. Because of this, the robot can physically move in the gaps it got stuck, but could not because of the way the robot was modelled.

With systems that have different communication mediums and multiple processes there is always a factor of delay in data transmission. This would mean if the robot is moving fast and near to an obstacle head on, stopping in time would be a problem. This factor was not accounted for in the obstacle avoidance constraint and only the kinematic model of the robot was considered. Having a dynamic model of the robot would help in a control structure that is highly responsive and efficient.

5.2 Discussion

In this thesis, we had three factors/research questions that formed the base of the research performed. These questions are highlighted again below:

1. How to include Human in the Loop control for shared control?
2. How can having a shared control approach help with more efficient robot control in terms of maneuverability?
3. Can the use of a shared control approach help with increasing the safety of robot during tele-operation?

We have created a human-in-the-loop control structure using an MPC approach. The current problem formulation allows for the user to constantly update the goal that is required for the MPC. This allows for the system to work in semi-autonomy providing obstacle avoidance. The system does not have any prior information of the final goal to be reached, but, only the goal poses updated from the human input at each time step. This give human the freedom to manoeuvre the robot at his/her will in the given environment. This is however, only one method on how the human inputs can be included into the control structure.

The implemented shared control approach was tested using user tests. These user tests help in providing a completely unbiased and neutral understanding of how the system performs. The user tests also help in reflecting to the second and third research questions.

Task efficiency in mobile robotics using completely autonomous systems has always been comparatively lower when compared to systems operated manually by humans. But concerns on safety have been higher with humans having to control machines remotely. Having a shared control approach will largely help in reducing these human errors. This can be seen from the results of the Task-1 of user testings performed in the figure 4.5.

Although the safety factor has been tended to using the implemented approach, it does not by large increase the task performance efficiency. While, an increase in efficiency is not achieved, we can see the from the test results in figure 4.6 that, task completion times are almost the same after with the benefit of having complete safety while performing the tasks. Using a shared control approach can also seem to be a bit of an overwhelming experience for some people. This means even with a shared control approach, the users would need a certain level of training period to be able to effectively operate the robot.

6 CONCLUSIONS

This thesis has presented the methodology and outcomes of the implementation of a shared control tele-operation strategy that enables a human-in-the-loop control structure, using Model Predictive Control. The developed control strategy allows a user to effectively manoeuvre a mobile robot within an environment with multiple obstacles while effectively reducing human errors that can transpire during tele-operation tasks. This has been the key motivation and objective of carrying out this research as part of the thesis.

The proposed shared control method is developed using Model Predictive Control as the base controller for the robot, enabling the robot to have a certain level of autonomy in terms of obstacle avoidance and goal attainment. The autonomous system although does not know its final goal, as is the case in unstructured environmental tasks like manoeuvring for search and rescue tasks[40] to undecided final locations that the user decides on the go. This drawback of a autonomous system is overcome using the proposes shared control method, allowing the human to control the desired position of the robot and not the robot directly.

The MPC problem formulated for the shared control approach is solved using Matlab and CasADi. The CasADi toolbox allows for quick solving of complex non-linear problems to allow as close to a real time computing as possible. The flexibility provided by CasADi to use different programming languages, allows for the design of the MPC controller to be developed either on the robot or remotely based on the user desires for the carrying out of tasks on a real robot. However, during this thesis the implementation could only be performed through simulations using the gazebo simulation environment and ROS.

The implementation was then tested not only by the author, but also through user tests enabling the author to come at an unbiased analysis of the implemented approach. The user tests, proved to be essential as the results from users of different experience levels and backgrounds varied a lot and the data from the tests can be seen in the appendix C.2. Although not the ideal performance of the chosen methodology was attained the improvement of the control strategy can be considered for further development of the research.

6.1 Future Scope

The implemented shared control approach though it provides a positive outcome to a couple of the research questions defined for this research, it is not a perfect solution. During the course of performing the research, most of the time was spent with choosing the right simulation environment for testing and the tools and software that will be used for the implementation of the thesis. There was not much time as planned initially for development of the proposed methodology and performing a more extensive user testing. This resulted in a lack of time for testing different shared control strategies and use cases to get a better idea on the benefits and importance of having a human-in-the-loop control structure as compared to using a manual tele-operation. There was also not sufficient time to test the short comings of using a completely autonomous system, and how shared control approaches could benefit in those situations. The tasks missed out on due to the impediments caused during this research can be carried out as future research works and are enlisted below:

1. Testing of the current shared control approach with modified MPC formulation tending to the shortcomings faced in this thesis.
2. Testing a few other shared control approaches
   (a) Hindsight Optimization.
   (b) Using of virtual fixtures.
   (c) Developing of a shared control strategy with haptic force feedback for better situational awareness.
3. Working in more complex environments with dynamic obstacles and researching other obstacle modelling methodologies.
4. Testing the implementation on real robots, and porting the proposed control algorithm to machines of different models and testing the flexibility in implementation of the shared control strategies.

REFERENCES

[1] Aarno, D., Ekvall, S. and Kragic, D. Adaptive Virtual Fixtures for Machine-Assisted Teleoperation Tasks. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. Apr. 2005, 1139–1144. DOI: 10.1109/ROBOT.2005.1570269.

[2] Abbink, D. A., Carlson, T., Mulder, M., de Winter, J. C. F., Aminravan, F., Gibo, T. L. and Boer, E. R. A Topology of Shared Control Systems—Finding Common Ground in Diversity. *IEEE Transactions on Human-Machine Systems* 48.5 (2018), 509–525.

[3] Abbott, J., Marayong, P. and Okamura, A. Haptic Virtual Fixtures for Robot-Assisted Manipulation. Vol. 28. Jan. 2005, 49–64. DOI: 10.1007/978-3-540-48113-3_5.

[4] Agrawal, S., Chen, X., Ragonesi, C. and Galloway, J. Training Toddlers Seated on Mobile Robots to Steer Using Force-Feedback Joystick. *Haptics, IEEE Transactions on* 5 (Jan. 2012), 376–383. DOI: 10.1109/TOH.2011.67.

[5] Akiba, S., Zanma, T. and Ishida, M. Optimal tracking control of two-wheeled mobile robots based on model predictive control. *2010 11th IEEE International Workshop on Advanced Motion Control (AMC)*. Mar. 2010, 454–459. DOI: 10.1109/AMC.2010.5464088.

[6] Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B. and Diehl, M. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation* 11 (2019), 1–36.

[7] Biyik, E. and Husein, M. Damping wide-area oscillations in power systems: A model predictive control design. *Turkish Journal of Electrical Engineering and Computer Sciences* 26 (Jan. 2018), 467–478. DOI: 10.3906/elk-1705-247.

[8] DFKI, t. G. R. C. f. A. I. *MiR configuration file for Gazebo Simulation*. URL: https://github.com/dfki-ric/mir_robot.

[9] Dragan, A. and Srinivasa, S. Assistive Teleoperation for Manipulation Tasks. *Proceedings of ACM/IEEE International Conference on Human-Robot Interaction (Late-Breaking Report)*. Mar. 2012.

[10] Erickson, Z., Clever, H., Turk, G., Liu, C. and Kemp, C. Deep Haptic Model Predictive Control for Robot-Assisted Dressing. May 2018, 1–8. DOI: 10.1109/ICRA.2018.8460656.

[11] Gao, M., Oberländer, J., Schamm, T. and Zöllner, J. M. Contextual task-aware shared autonomy for assistive mobile robot teleoperation. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, 3311–3318. DOI: 10.1109/IROS.2014.6943023.

[12] Goodrich, M. and Schultz, A. Human-Robot Interaction: A Survey. *Foundations and Trends in Human-Computer Interaction* 1 (Jan. 2007), 203–275. DOI: 10.1561/1100000005.

[13] Herlant, L. V., Holladay, R. M. and Srinivasa, S. S. Assistive teleoperation of robot arms via automatic time-optimal mode switching. *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 2016, 35–42.

[14] Holkar, K. and Waghmare, L. An Overview of Model Predictive Control. *International Journal of Control and Automation* 3.4 (Dec. 2010), 47–64.

[15] Javdani, S., Admoni, H., Pellegrinelli, S., Srinivasa, S. S. and Bagnell, J. A. Shared autonomy via hindsight optimization for teleoperation and teaming. *The International Journal of Robotics Research* 37.7 (2018), 717–742. DOI: 10.1177/0278364918776060.

[16] Kaber, D. B. and Endsley, M. R. The effects of level of automation and adaptive automation on human performance, situation awareness and workload in a dynamic control task. *Theoretical Issues in Ergonomics Science* 5.2 (2004), 113–153. DOI: 10.1080/1463922021000054335.

[17] Kam, H. R., Lee, S.-H., Park, T. and Kim, C.-H. RViz: A Toolkit for Real Domain Data Visualization. *Telecommun. Syst.* 60.2 (Oct. 2015), 337–345. ISSN: 1018-4864. DOI: 10.1007/s11235-015-0034-5. URL: https://doi.org/10.1007/s11235-015-0034-5.

[18] Kanjanawanishkul, K. and Zell, A. Path following for an omnidirectional mobile robot based on model predictive control. *2009 IEEE International Conference on Robotics and Automation*. May 2009, 3341–3346. DOI: 10.1109/ROBOT.2009.5152217.

[19] Koenig, N. and Howard, A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. Sept. 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[20] Kontz, M., Beckwith, J. and Book, W. Evaluation of a Teleoperated Haptic Forklift. Vol. 1. Feb. 2005, 295–300. ISBN: 0-7803-9047-4. DOI: 10.1109/AIM.2005.1501006.

[21] Li, Z., Yang, C., Su, C., Deng, J. and Zhang, W. Vision-Based Model Predictive Control for Steering of a Nonholonomic Mobile Robot. *IEEE Transactions on Control Systems Technology* 24.2 (Mar. 2016), 553–564. ISSN: 1558-0865. DOI: 10.1109/TCST.2015.2454484.

[22] Liang, J., Yu, G. and Guo, L. Human-Robot Collaborative Semi-Autonomous Teleoperation with Force Feedback. *2018 5th International Conference on Soft Computing Machine Intelligence (ISCMI)*. Nov. 2018, 129–134. DOI: 10.1109/ISCMI.2018.8703223.

[23] *Logitech Attack 3 Joystick, from Logitech*. URL: https://support.logi.com/hc/en-in/articles/360024319493.

[24] Losey, D., McDonald, C., Battaglia, E. and O'Malley, M. A review of intent detection, arbitration, and communication aspects of shared control for physical human-robot interaction. *Applied Mechanics Reviews* 70 (Jan. 2018). DOI: 10.1115/1.4039145.

[25] Mars, F., Deroo, M. and Charron, C. Driver adaptation to haptic shared control of the steering wheel. *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2014, 1505–1509.

[26] Mars, F., Deroo, M. and Hoc, J. Analysis of Human-Machine Cooperation When Driving with Different Degrees of Haptic Shared Control. *IEEE Transactions on Haptics* 7.3 (July 2014), 324–333. ISSN: 2329-4051. DOI: 10.1109/TOH.2013.2295095.

[27] *MiR100 Mobile Robot*. URL: https://www.mobile-industrial-robots.com/en/solutions/robots/mir100/.

[28] Moffitt, V., Franke, J. and Lomas, M. Mixed-initiative adjustable autonomy in multi-vehicle operations. (Jan. 2006).

[29] Murphy, R. R., Dreger, K. L., Newsome, S., Rodocker, J., Steimle, E., Kimura, T., Makabe, K., Matsuno, F., Tadokoro, S. and Kon, K. Use of remotely operated marine vehicles at Minamisanriku and Rikuzentakata Japan for disaster recovery. *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*. Nov. 2011, 19–25. DOI: 10.1109/SSRR.2011.6106798.

[30] Nadrag, P., Temzi, L., Arioui, H. and Hoppenot, P. Remote control of an assistive robot using force feedback. *2011 15th International Conference on Advanced Robotics (ICAR)*. June 2011, 211–216. DOI: 10.1109/ICAR.2011.6088570.

[31] Nunes, G. C. Design and analysis of multivariable predictive control applied to an oil-water-gas separator: A polynomial approach. PhD thesis. Citeseer, 2001.

[32] Ostafew, C., Schoellig, A. and Barfoot, T. Learning-based nonlinear model predictive control to improve vision-based mobile robot path-tracking in challenging outdoor environments. *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, 4029–4036. DOI: 10.1109/ICRA.2014.6907444.

[33] Ostafew, C., Schoellig, A. and Barfoot, T. Robust Constrained Learning-based NMPC enabling reliable mobile robot path tracking. *The International Journal of Robotics Research* 35 (May 2016). DOI: 10.1177/0278364916645661.

[34] Pruks, V., Lee, K. and Ryu, J. Shared Teleoperation for Nuclear Plant Robotics Using Interactive Virtual Guidance Generation and Shared Autonomy Approaches. *2018 15th International Conference on Ubiquitous Robots (UR)*. June 2018, 91–95. DOI: 10.1109/URAI.2018.8441814.

[35] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R. and Ng, A. ROS: an open-source Robot Operating System. *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*. Kobe, Japan, May 2009.

[36] Sellner, B., Hiatt, L., Simmons, R. and Singh, S. Attaining situational awareness for sliding autonomy. *Proceedings of HRI 2006*. 2006, 80–87. DOI: 10.1145/1121241.1121257.

[37] Shah, S., Dey, D., Lovett, C. and Kapoor, A. *AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles*. 2017. arXiv: 1705.05065 [cs.RO].

[38] Song, L., Guo, H., Wang, F., Liu, J. and Chen, H. Model predictive control oriented shared steering control for intelligent vehicles. *2017 29th Chinese Control And Decision Conference (CCDC)*. May 2017, 7568–7573. DOI: 10.1109/CCDC.2017.7978557.

[39] Storms, J. G. and Tilbury, D. M. Blending of human and obstacle avoidance control for a high speed mobile robot. *2014 American Control Conference*. June 2014, 3488–3493. DOI: 10.1109/ACC.2014.6859352.

[40] Storms, J., Chen, K. and Tilbury, D. A shared control method for obstacle avoidance with mobile robots and its interaction with communication delay. *The International Journal of Robotics Research* 36.5-6 (2017), 820–839. DOI: 10.1177/0278364917693690.

[41] Tsoi, K. K., Mulder, M. and Abbink, D. A. Balancing safety and support: Changing lanes with a haptic lane-keeping support system. *2010 IEEE International Conference on Systems, Man and Cybernetics*. 2010, 1236–1243.

[42] Vazquez, S., Leon, J. I., Franquelo, L. G., Rodriguez, J., Young, H. A., Marquez, A. and Zanchetta, P. Model predictive control: A review of its applications in power electronics. *IEEE industrial electronics magazine* 8.1 (2014), 16–31.

[43] Worthmann, K., Mehrez, M., Zanon, M., Mann, G. K., Gosine, R. and Diehl, M. Regulation of Differential Drive Robots using Continuous Time MPC without Stabilizing Constraints or Costs. *IFAC2015-PapersOnLine* 48.23 (Dec. 2015), 129–135.

[44] Yan, J., Yang, C., Lu, Q. and Zhao, J. Virutual guides for force feedback teleoperation. *2013 IEEE International Conference on Information and Automation (ICIA)*. 2013, 145–150. DOI: 10.1109/ICInfA.2013.6720286.

[45] Yu, N., Wang, K., Li, Y., Xu, C. and Liu, J. A haptic shared control approach to teleoperation of mobile robots. *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*. June 2015, 31–35. DOI: 10.1109/CYBER.2015.7287905.

[46] Zeestraten, M. J. A., Havoutis, I. and Calinon, S. Programming by Demonstration for Shared Control With an Application in Teleoperation. *IEEE Robotics and Automation Letters* 3.3 (July 2018), 1848–1855. ISSN: 2377-3766. DOI: 10.1109/LRA.2018.2805105.

[47] Zhengcai Cao, Yingtao Zhao and Shuguo Wang. Trajectory tracking and point stabilization of nonholonomic mobile robot. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2010, 1328–1333. DOI: 10.1109/IROS.2010.5650385.

A MODELS

A.1 The modified MiR Description File

```
1  <?xml version="1.0"?>
2  <robot xmlns:xacro="http://ros.org/wiki/xacro">
3    <xacro:include filename="$(find␣mir_description)/urdf/include/
       common_properties.urdf.xacro" />
4    <xacro:include filename="$(find␣mir_description)/urdf/include/imu.
       gazebo.urdf.xacro" />
5    <xacro:include filename="$(find␣mir_description)/urdf/include/mir_100
       .gazebo.xacro" />
6    <xacro:include filename="$(find␣mir_description)/urdf/include/mir_100
       .transmission.xacro" />
7    <xacro:include filename="$(find␣mir_description)/urdf/include/
       sick_s300.urdf.xacro" />
8    <xacro:include filename="$(find␣mir_description)/urdf/include/
       camera_mir.urdf.xacro" />
9
10   <xacro:property name="deg_to_rad" value="0.017453293" />
11
12   <xacro:property name="mir_100_base_mass" value="58.0" />
13
14   <xacro:property name="mir_100_act_wheel_radius" value="0.0625" />
15   <xacro:property name="mir_100_act_wheel_width" value="0.032" />
16   <xacro:property name="mir_100_act_wheel_mass" value="1.0" />
17   <xacro:property name="mir_100_act_wheel_dx" value="0.037646" />
18   <xacro:property name="mir_100_act_wheel_dy" value="0.222604" />
19
20   <xacro:property name="mir_100_caster_wheel_radius" value="${
       mir_100_act_wheel_radius}" />
21   <xacro:property name="mir_100_caster_wheel_width" value="${
       mir_100_act_wheel_width}" />
22   <xacro:property name="mir_100_caster_wheel_mass" value="${
       mir_100_act_wheel_mass}" />
23   <xacro:property name="mir_100_caster_wheel_dx" value="-0.0382" />
```

```
24    <xacro:property name="mir_100_caster_wheel_dy" value="0" />
25    <xacro:property name="mir_100_caster_wheel_dz" value="-0.094" />
26    <xacro:property name="mir_100_front_caster_wheel_base_dx" value="
          0.341346" />
27    <xacro:property name="mir_100_back_caster_wheel_base_dx" value="
          -0.270154" />
28    <xacro:property name="mir_100_caster_wheel_base_dy" value="0.203" />
29    <xacro:property name="mir_100_caster_wheel_base_dz" value="${
          mir_100_caster_wheel_radius-mir_100_caster_wheel_dz}" />
30
31    <xacro:property name="imu_stdev" value="0.00017" />
32
33    <xacro:macro name="actuated_wheel" params="prefix locationprefix
          locationright">
34      <joint name="${prefix}${locationprefix}_wheel_joint" type="
            continuous">
35        <origin xyz="0.0 ${-mir_100_act_wheel_dy * locationright} ${
              mir_100_act_wheel_radius}" rpy="0 0 0" />
36        <parent link="${prefix}base_link" />
37        <child link="${prefix}${locationprefix}_wheel_link" />
38        <axis xyz="0 1 0" />
39        <limit effort="100" velocity="20.0" />
40      </joint>
41
42      <link name="${prefix}${locationprefix}_wheel_link">
43        <xacro:cylinder_inertial mass="${mir_100_act_wheel_mass}" radius=
              "${mir_100_act_wheel_radius}" length="${
              mir_100_act_wheel_width}">
44          <origin xyz="0 0 0" rpy="${0.5 * pi} 0 0" />
45        </xacro:cylinder_inertial>
46        <visual>
47          <origin xyz="0 0 0" rpy="0 0 0" />
48          <geometry>
49            <mesh filename="package://mir_description/meshes/visual/wheel
                  .stl" />
50          </geometry>
51          <xacro:insert_block name="material_dark_grey" />
52        </visual>
53        <collision>
54          <origin xyz="0 0 0" rpy="0 0 0" />
55          <geometry>
56            <mesh filename="package://mir_description/meshes/visual/wheel
```

```
                .stl" />
57            </geometry>
58          </collision>
59        </link>
60        <gazebo reference="${prefix}${locationprefix}_wheel_link">
61          <material>Gazebo/DarkGrey</material>
62        </gazebo>
63      </xacro:macro>
64
65      <xacro:macro name="caster_wheel" params="prefix␣locationprefix␣
          locationright␣wheel_base_dx">
66        <!-- caster hub -->
67        <joint name="${prefix}${locationprefix}_caster_rotation_joint" type
            ="continuous">
68          <origin xyz="${wheel_base_dx}␣${-mir_100_caster_wheel_base_dy␣*␣
              locationright}␣${mir_100_caster_wheel_base_dz}" rpy="0␣0␣0" />
69          <parent link="${prefix}base_link" />
70          <child link="${prefix}${locationprefix}_caster_rotation_link" />
71          <axis xyz="0␣0␣1" />
72          <dynamics damping="0.01" friction="0.0"/>
73        </joint>
74
75        <link name="${prefix}${locationprefix}_caster_rotation_link">
76          <inertial>
77            <!-- <origin xyz="0␣0␣-0.042500000044" rpy="${0.5␣*␣pi}␣${24␣*␣
                deg_to_rad}␣${1.5␣*␣pi}" /> -->
78            <origin xyz="0␣0␣-0.042500000044" rpy="${24␣*␣deg_to_rad}␣0␣
                ${0.5␣*␣pi}␣" />
79            <mass value="0.3097539019" />
80            <inertia
81              ixx="0.0005844517978"
82              ixy="0"
83              ixz="0"
84              iyy="0.00052872551237"
85              iyz="0"
86              izz="0.00017923555074" />
87          </inertial>
88          <visual>
89            <origin xyz="0␣0␣0" rpy="0␣0␣0" />
90            <geometry>
91              <mesh filename="package://mir_description/meshes/visual/
                    caster_wheel_base.stl" />
```

```
92          </geometry>
93          <xacro:insert_block name="material_silver" />
94        </visual>
95        <collision>
96          <origin xyz="0␣0␣0" rpy="0␣0␣0" />
97          <geometry>
98            <mesh filename="package://mir_description/meshes/collision/
                caster_wheel_base.stl" />
99          </geometry>
100       </collision>
101     </link>
102     <gazebo reference="${prefix}${locationprefix}_caster_rotation_link"
          >
103       <material>Gazebo/Grey</material>
104     </gazebo>
105
106     <!-- caster wheel -->
107     <joint name="${prefix}${locationprefix}_caster_wheel_joint" type="
          continuous">
108       <origin xyz="${mir_100_caster_wheel_dx}␣${-
            mir_100_caster_wheel_dy␣*␣locationright}␣${
            mir_100_caster_wheel_dz}" rpy="0␣0␣0" />
109       <parent link="${prefix}${locationprefix}_caster_rotation_link" />
110       <child link="${prefix}${locationprefix}_caster_wheel_link" />
111       <axis xyz="0␣1␣0" />
112     </joint>
113
114     <link name="${prefix}${locationprefix}_caster_wheel_link">
115       <xacro:cylinder_inertial mass="${mir_100_caster_wheel_mass}"
            radius="${mir_100_caster_wheel_radius}" length="${
            mir_100_caster_wheel_width}">
116         <origin xyz="0␣0␣0" rpy="${0.5␣*␣pi}␣0␣0" />
117       </xacro:cylinder_inertial>
118       <visual>
119         <origin xyz="0␣0␣0" rpy="0␣0␣0" />
120         <geometry>
121           <mesh filename="package://mir_description/meshes/visual/wheel
                .stl" />
122         </geometry>
123         <xacro:insert_block name="material_dark_grey" />
124       </visual>
125       <collision>
```

```
126         <origin xyz="0␣0␣0" rpy="0␣0␣0" />
127         <geometry>
128           <mesh filename="package://mir_description/meshes/visual/wheel
                .stl" />
129         </geometry>
130       </collision>
131     </link>
132     <gazebo reference="${prefix}${locationprefix}_caster_wheel_link">
133       <material>Gazebo/DarkGrey</material>
134     </gazebo>
135   </xacro:macro>
136
137   <xacro:macro name="mir_100" params="prefix">
138     <link name="${prefix}base_footprint" />
139
140     <joint name="${prefix}base_joint" type="fixed">
141       <parent link="${prefix}base_footprint" />
142       <child link="${prefix}base_link" />
143       <origin xyz="0␣0␣0" rpy="0␣0␣0" />
144     </joint>
145
146     <link name="${prefix}base_link">
147       <xacro:box_inertial mass="${mir_100_base_mass}" x="0.9" y="0.58"
            z="0.3">
148         <origin xyz="${mir_100_act_wheel_dx}␣0␣0.20" rpy="0␣0␣0" />
149       </xacro:box_inertial>
150       <visual>
151         <origin xyz="${mir_100_act_wheel_dx}␣0␣0" rpy="0␣0␣0" />
152         <geometry>
153           <mesh filename="package://mir_description/meshes/visual/
                mir_100_base.stl" />
154         </geometry>
155         <xacro:insert_block name="material_white" />
156       </visual>
157       <collision>
158         <origin xyz="${mir_100_act_wheel_dx}␣0␣0" rpy="0␣0␣0" />
159         <geometry>
160           <mesh filename="package://mir_description/meshes/collision/
                mir_100_base.stl" />
161         </geometry>
162       </collision>
163     </link>
```

```
164     <gazebo reference="${prefix}base_link">
165       <material>Gazebo/White</material>
166     </gazebo>
167
168     <!-- IMU -->
169     <joint name="${prefix}base_link_to_imu_joint" type="fixed">
170       <parent link="${prefix}base_link" />
171       <child link="${prefix}imu_link" />
172       <origin xyz="0.0 0.0 0.25" rpy="0 0 0" /> <!-- same as real MiR
              -->
173     </joint>
174
175     <link name="${prefix}imu_link" />
176
177     <xacro:imu_gazebo link="${prefix}imu_link" imu_topic="imu_data"
          update_rate="50.0" stdev="${imu_stdev}" />
178
179     <!-- Create an alias for imu_link. This is necessary because the
          real MiR's
180 ␣␣␣␣␣␣␣␣TF␣has␣imu_link,␣but␣the␣imu_data␣topic␣is␣published␣in␣the␣
      imu_frame
181 ␣␣␣␣␣␣␣␣frame.␣-->
182 ␣␣␣␣<joint␣name="${prefix}imu_link_to_imu_frame_joint"␣type="fixed">
183 ␣␣␣␣␣␣<parent␣link="${prefix}imu_link"␣/>
184 ␣␣␣␣␣␣<child␣link="${prefix}imu_frame"␣/>
185 ␣␣␣␣␣␣<origin␣xyz="0␣0␣0"␣rpy="0␣0␣0"␣/>
186 ␣␣␣␣</joint>
187
188 ␣␣␣␣<link␣name="${prefix}imu_frame"␣/>
189
190 ␣␣␣␣<!--␣Laser␣scanners␣-->
191 ␣␣␣␣<joint␣name="${prefix}base_link_to_front_laser_joint"␣type="fixed">
192 ␣␣␣␣␣␣<parent␣link="${prefix}base_link"␣/>
193 ␣␣␣␣␣␣<child␣link="${prefix}front_laser_link"␣/>
194 ␣␣␣␣␣␣<origin␣xyz="0.4288␣0.2358␣0.1914"␣rpy="0.0␣0.0␣${0.25␣*␣pi}"␣/>␣
      ␣<!--␣from␣visually␣matching␣up␣the␣meshes␣of␣the␣MiR␣and␣the␣laser␣
      scanner␣-->
195 ␣␣␣␣</joint>
196 ␣␣␣␣<xacro:sick_s300␣prefix="${prefix}"␣link="front_laser_link"␣topic="
      f_scan"␣/>
197
198 ␣␣␣␣<joint␣name="${prefix}base_link_to_back_laser_joint"␣type="fixed">
```

```
199 ␣␣␣␣␣␣<parent␣link="${prefix}base_link"␣/>
200 ␣␣␣␣␣␣<child␣link="${prefix}back_laser_link"␣/>
201 ␣␣␣␣␣␣<origin␣xyz="-0.3548␣-0.2352␣0.1914"␣rpy="0.0␣0.0␣${-0.75␣*␣pi}"␣
    />␣␣<!--␣from␣visually␣matching␣up␣the␣meshes␣of␣the␣MiR␣and␣the␣
    laser␣scanner␣-->
202 ␣␣␣␣</joint>
203
204 ␣␣␣␣<xacro:sick_s300␣prefix="${prefix}"␣link="back_laser_link"␣topic="
    b_scan"␣/>
205
206 ␣␣␣␣␣␣␣␣<!--␣Cameras␣-->␣␣␣␣␣␣␣␣
207 ␣␣␣␣␣␣␣␣<joint␣name="${prefix}camera_joint_forward"␣type="fixed">
208 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<parent␣link="${prefix}base_link"/>
209 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<child␣link="${prefix}front_camera_link"/>
210 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<axis␣xyz="0␣1␣0"␣/>
211 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<origin␣xyz="0.4␣0␣0.42"␣rpy="0␣0␣0"/>
212 ␣␣␣␣␣␣␣␣</joint>
213 ␣␣␣␣␣␣␣␣<xacro:camera_mir␣prefix="${prefix}"␣link="front_camera_link"␣
    />
214
215 ␣␣␣␣␣␣␣␣<joint␣name="${prefix}camera_joint_back"␣type="fixed">
216 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<parent␣link="${prefix}base_link"/>
217 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<child␣link="${prefix}back_camera_link"/>
218 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<axis␣xyz="0␣1␣0"␣/>
219 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<origin␣xyz="-0.34␣0␣0.42"␣rpy="0␣0␣3.14"/>
220 ␣␣␣␣␣␣␣␣</joint>
221 ␣␣␣␣␣␣␣␣<xacro:camera_mir␣prefix="${prefix}"␣link="back_camera_link"␣/>
222
223
224 ␣␣␣␣<!--␣Ultrasound␣sensors␣-->
225 ␣␣␣␣<joint␣name="${prefix}us_1_joint"␣type="fixed">␣␣␣<!--␣right␣
    ultrasound␣-->
226 ␣␣␣␣␣␣<parent␣link="${prefix}base_link"␣/>
227 ␣␣␣␣␣␣<child␣link="${prefix}us_1_frame"␣/>
228 ␣␣␣␣␣␣<origin␣xyz="0.45␣-0.12␣0.16␣"␣rpy="0␣0␣0"␣/>␣␣<!--␣from␣visually
    ␣matching␣to␣the␣mesh␣of␣the␣MiR␣-->
229 ␣␣␣␣</joint>
230
231 ␣␣␣␣<link␣name="${prefix}us_1_frame"␣/>
232
233 ␣␣␣␣<joint␣name="${prefix}us_2_joint"␣type="fixed">␣␣␣<!--␣left␣
    ultrasound␣-->
```

```
234 ␣␣␣␣␣␣<parent␣link="${prefix}base_link"␣/>
235 ␣␣␣␣␣␣<child␣link="${prefix}us_2_frame"␣/>
236 ␣␣␣␣␣␣<origin␣xyz="0.45␣0.12␣0.16␣"␣rpy="0␣0␣0"␣/>␣␣<!--␣from␣visually␣
        matching␣to␣the␣mesh␣of␣the␣MiR␣-->
237 ␣␣␣␣</joint>
238
239 ␣␣␣␣<link␣name="${prefix}us_2_frame"␣/>
240
241
242 ␣␣␣␣<!--␣wheels␣-->
243 ␣␣␣␣<xacro:actuated_wheel␣prefix="${prefix}"␣locationprefix="left"␣
        locationright="-1"/>
244 ␣␣␣␣<xacro:actuated_wheel␣prefix="${prefix}"␣locationprefix="right"␣
        locationright="1"/>
245 ␣␣␣␣<xacro:caster_wheel␣prefix="${prefix}"␣locationprefix="fl"␣
        locationright="-1"␣wheel_base_dx="${
        mir_100_front_caster_wheel_base_dx}"/>
246 ␣␣␣␣<xacro:caster_wheel␣prefix="${prefix}"␣locationprefix="fr"␣
        locationright="1"␣wheel_base_dx="${
        mir_100_front_caster_wheel_base_dx}"/>
247 ␣␣␣␣<xacro:caster_wheel␣prefix="${prefix}"␣locationprefix="bl"␣
        locationright="-1"␣wheel_base_dx="${
        mir_100_back_caster_wheel_base_dx}"/>
248 ␣␣␣␣<xacro:caster_wheel␣prefix="${prefix}"␣locationprefix="br"␣
        locationright="1"␣wheel_base_dx="${mir_100_back_caster_wheel_base_dx
        }"/>
249
250 ␣␣␣␣<joint␣name="${prefix}base_link_surface_joint"␣type="fixed">
251 ␣␣␣␣␣␣<origin␣xyz="${mir_100_act_wheel_dx}␣0␣0.352"␣rpy="0␣0␣0"␣/>
252 ␣␣␣␣␣␣<parent␣link="${prefix}base_link"␣/>
253 ␣␣␣␣␣␣<child␣link="${prefix}surface"␣/>
254 ␣␣␣␣␣␣<axis␣xyz="0␣0␣1"␣/>
255 ␣␣␣␣</joint>
256
257 ␣␣␣␣<link␣name="${prefix}surface"/>
258
259 ␣␣␣␣<xacro:mir_100_wheel_transmissions␣prefix="${prefix}"/>
260
261 ␣␣␣␣<!--␣set␣the␣gazebo␣friction␣parameters␣for␣the␣wheels␣-->
262 ␣␣␣␣<xacro:set_all_wheel_frictions␣prefix="${prefix}"/>
263
264 ␣␣␣␣<p3d_base_controller␣prefix="${prefix}"␣/>
```

```
  </xacro:macro>
</robot>
```

Annex A.1. *The modified MiR description file to include cameras.*

A.2 The camera description file

```
<?xml version="1.0"?>

<robot xmlns:xacro="http://ros.org/wiki/xacro">

  <xacro:macro name="camera_mir" params="link_prefix">
  <xacro:include filename="$(find mir_description)/urdf/include/
      common_properties.urdf.xacro" />


    <link name="${prefix}${link}">
        <collision>
          <origin xyz="0 0 0" rpy="0 0 0"/>
          <geometry>
                <box size="0.05 0.1 0.025"/>
          </geometry>
        </collision>

        <visual>
          <origin xyz="0 0 0" rpy="0 0 0"/>
          <geometry>
                <box size="0.05 0.1 0.025"/>
          </geometry>
          <material name="yellow">
                <color rgba="${255/255} ${226/255} ${0/255} 1"/>
          </material>
        </visual>

        <inertial>
                <mass value="0.001" />
                <origin xyz="0 0 0" rpy="0 0 0" />
                <inertia ixx="0.0001" ixy="0" ixz="0" iyy="0.0001" iyz=
                    "0" izz="0.0001" />
        </inertial>

    </link>

```

```
35      <gazebo reference="${prefix}${link}">
36        <sensor type="camera" name="${prefix}${link}">
37          <update_rate>30.0</update_rate>
38          <camera>
39            <pose>0.0 0.0 0.0 0.0 0.0 0.0</pose>
40            <horizontal_fov>${2.4}</horizontal_fov>
41            <image>
42              <format>R8G8B8</format>
43              <width>1020</width>
44              <height>1020</height>
45            </image>
46            <clip>
47              <near>0.01</near>
48              <far>100</far>
49            </clip>
50          </camera>
51
52          <plugin name="camera_${link}_controller" filename="
                libgazebo_ros_camera.so">
53            <alwaysOn>true</alwaysOn>
54            <updateRate>0.0</updateRate>
55            <cameraName>${prefix}/camera/${link}</cameraName>
56            <imageTopicName>image_raw</imageTopicName>
57            <cameraInfoTopicName>camera_info</cameraInfoTopicName>
58            <frameName>${link}</frameName>
59            <hackBaseline>0.07</hackBaseline>
60            <distortionK1>0.0</distortionK1>
61            <distortionK2>0.0</distortionK2>
62            <distortionK3>0.0</distortionK3>
63            <distortionT1>0.0</distortionT1>
64            <distortionT2>0.0</distortionT2>
65          </plugin>
66        </sensor>
67      </gazebo>
68    </xacro:macro>
69
70  </robot>
```

Annex A.2. *Camera definition URDF file used by the modified MiR description file.*

B CODES

B.1 Matlab MPC Code - no obstacle Matlab simulation

```
clear all
clc

%
% CasADi v3.4.5
addpath('D:\casadi-windows-matlabR2016a-v3.5.5')
import casadi.*


%
N =15; % prediction horizon

v_max = 0.8; v_min = -v_max;
omega_max = 1; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);

% Kinematic System Model r.h.s

rhs = [v*cos(theta);v*sin(theta);omega];

 % nonlinear mapping function f(x,u)

f = Function('f',{states,controls},{rhs});
% Decision variables (controls)
U = SX.sym('U',n_controls,N);
% Decision Variables (states)
X = SX.sym('X',n_states,(N+1));
```

```
33 % parameters (initial state of the robot and the goal state)
34 P = SX.sym('P',n_states + n_states);
35
36 obj = 0; % Objective function
37 g = [];  % constraints vector
38
39 % Weighting Matrices
40 Q = zeros(3,3); Q(1,1) = 3;Q(2,2) = 3;Q(3,3) = 1.5;
41 R = zeros(2,2); R(1,1) = 0.5; R(2,2) = 0.5;
42
43 % Symbolic computation of constraints and objective function
44 st  = X(:,1); % initial state
45 g = [g;st-P(1:3)]; % initial condition constraints
46
47 T = 0.25; % dt in Seconds
48 % Define the equality constraints as symbolics
49 for k = 1:N
50     st = X(:,k);  con = U(:,k);
51     obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
52     st_next = X(:,k+1);
53     f_value = f(st,con);
54     st_next_euler = st+ (T*f_value);
55     g = [g;st_next-st_next_euler]; % compute constraints
56 end
57
58 vp_lims = 0.1;
59 vn_lims = 0.1;
60 wp_lims = 0.1;
61 wn_lims = 0.1;
62 for k = 1:N-1
63     g = [g ; (U(1,k)-U(1,k+1))];
64 end
65
66 for k = 1:N-1
67     g = [g ; (U(2,k)-U(2,k+1))];
68 end
69
70 %
71 % make the decision variable one column vector
72 OPT_variables = [reshape(X,3*(N+1),1);reshape(U,2*N,1)];
73
74 nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
```

```
75
76 opts = struct;
77 opts.ipopt.max_iter = 10000;
78 opts.ipopt.print_level =0;%0,3
79 opts.print_time = 0;
80 opts.ipopt.acceptable_tol =1e-8;
81 opts.ipopt.acceptable_obj_change_tol = 1e-6;
82
83 solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
84
85 args = struct;
86 % equality constraints
87 args.lbg(1:3*(N+1)) = 0;
88 args.ubg(1:3*(N+1)) = 0;
89
90 length_lin_const = 3*(N+1)+(N-1);
91 % linear velocity decomposition constraints
92 args.lbg(3*(N+1)+1 : length_lin_const) = -vn_lims;
93 args.ubg(3*(N+1)+1 : length_lin_const) = vp_lims;
94
95 length_vp = length_lin_const+(N-1);
96 % angular velocity decomposition constraints
97 args.lbg(length_lin_const+1 : length_vp) = -wn_lims;
98 args.ubg(length_lin_const+1 : length_vp) = wp_lims;
99
100 % Bounds on the state variables (Boundary Conditions/Outer walls)
101 args.lbx(1:3:3*(N+1),1) = -4.2; %state x lower bound
102 args.ubx(1:3:3*(N+1),1) = 4.2; %state x upper bound
103 args.lbx(2:3:3*(N+1),1) = -4.2; %state y lower bound
104 args.ubx(2:3:3*(N+1),1) = 4.2; %state y upper bound
105 args.lbx(3:3:3*(N+1),1) = -inf; %state theta lower bound
106 args.ubx(3:3:3*(N+1),1) = inf; %state theta upper bound
107
108 % Bounds on the control Variables (Linear and angular velocity)
109 args.lbx(3*(N+1)+1:2:3*(N+1)+2*N,1) = -0.4; %v lower bound
110 args.ubx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_max; %v upper bound
111 args.lbx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_min; %omega lower bound
112 args.ubx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_max; %omega upper bound
113
114 %
115 prompt = {'Enter␣x-Value:','Enter␣y-Value:', 'Enter␣theta-Value␣(Rad)'};
116 dlgtitle = 'Initial␣estimate';
```

```
117 dims = [1 35];
118 definput = {'-4.0','-4.0', '0'};
119 answer = inputdlg(prompt,dlgtitle,dims,definput);
120
121 x_initial = str2double(answer{1,1});
122 y_initial = str2double(answer{2,1});
123 theta_initial = str2double(answer{3,1});
124
125 % Initial condition
126 x0 = [x_initial ; y_initial ; theta_initial];
127
128 prompt = {'Enter x-Value:','Enter y-Value:', 'Enter theta-Value (Rad)'};
129 dlgtitle = 'Goal Position';
130 dims = [1 35];
131 definput = {'4','4', '0'};
132 answer = inputdlg(prompt,dlgtitle,dims,definput);
133
134 x_final = str2double(answer{1,1});
135 y_final = str2double(answer{2,1});
136 theta_final = str2double(answer{3,1});
137
138 % Reference posture.
139 xs = [x_final ; y_final ; theta_final]; % Reference posture.
140
141 t0 = 0;
142 t(1) = t0;
143 u0 = zeros(N,2);        % two control inputs for each robot
144 X0 = repmat(x0,1,N+1)'; % initialization of the states decision variables
145
146 sim_tim = 30; % Maximum simulation time
147
148 % Start MPC
149 mpciter = 0;
150 %
151 main_loop = tic;
152 while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)
153     args.p   = [x0;xs]; % set the values of the parameters vector
154     % initial value of the optimization variables
155     args.x0  = [reshape(X0',3*(N+1),1);reshape(u0',2*N,1)];
156     % Run the NLP solver
157     sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
158         'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
```

```
159
160     % get controls only N+1 only
161     u = reshape(full(sol.x(3*(N+1)+1:end))',2,N)';
162
163     t(mpciter+1) = t0;
164
165     % Apply the control and shift the solution
166     [t0, x0, u0] = shift(T, t0, x0, u,f);
167
168     % Shift trajectory to initialize the next step
169     X0 = reshape(full(sol.x(1:3*(N+1)))',3,N+1)';
170     X0 = [X0(2:end,:);X0(end,:)];
171
172     mpciter = mpciter + 1;
173 end
174 main_loop_time = toc(main_loop);
175 ss_error = norm((x0-xs),2)
176 average_mpc_time = main_loop_time/(mpciter+1)
```

Annex B.1. *Matlab script for MPC adapted from [43]*

B.2 Matlab MPC Code - obstacle avoidance Matlab simulation

```
1  clear all
2  % close all
3  clc
4
5  %
6  % CasADi v3.4.5
7  addpath('D:\casadi-windows-matlabR2016a-v3.5.5')
8  import casadi.*
9
10
11 %
12 N =15; % prediction horizon
13 rob_diam = 0.9;
14
15 v_max = 0.8; v_min = -v_max;
16 omega_max = 1; omega_min = -omega_max;
17
18 x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
19 states = [x;y;theta]; n_states = length(states);
20
```

```
21 v = SX.sym('v'); omega = SX.sym('omega');
22 controls = [v;omega]; n_controls = length(controls);
23
24 % Kinematic System Model r.h.s
25
26 rhs = [v*cos(theta);v*sin(theta);omega];
27
28 f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
29 % Decision variables (controls)
30 U = SX.sym('U',n_controls,N);
31 % Decision Variables (states)
32 X = SX.sym('X',n_states,(N+1));
33 % parameters (initial state of the robot and the reference state)
34 P = SX.sym('P',n_states + n_states);
35
36
37 obj = 0; % Objective function
38 g = [];  % constraints vector
39
40 % Weighting Matrices
41 Q = zeros(3,3); Q(1,1) = 6;Q(2,2) = 6;Q(3,3) = 4;
42 R = zeros(2,2); R(1,1) = 0.3; R(2,2) = 0.15;
43
44 % Symbolic computation of constraints and objective function
45 st  = X(:,1); % initial state
46 g = [g;st-P(1:3)]; % initial condition constraints
47
48 T = 0.25; % dt in Seconds
49 % Define the equality constraints as symbolics
50 for k = 1:N
51     st = X(:,k);  con = U(:,k);
52     obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
53     st_next = X(:,k+1);
54     f_value = f(st,con);
55     st_next_euler = st+ (T*f_value);
56     g = [g;st_next-st_next_euler]; % compute constraints
57 end
58
59
60 % Add constraints for collision avoidance (Inequality constraint)
61
62 obs_x = [-4 -2 0.5 3 -2.5 0 2 -4 -1 3 -2 1 4 -1 1 3]; % meters
```

```
63 obs_y = [3.5 3 3.5 4 1.5 1.5 2 0 0 0.5 -2 -1 -1 -4 -3 -3]; % meters
64
65 obs_diam = 0.45; % meters
66 epsilon_= 1.0;
67 for k = 1:N+1
68     for num_obs = 1:length(obs_x)
69         g = [g ; -sqrt((X(1,k)-obs_x(num_obs))^2+(X(2,k)-...
70             obs_y(num_obs))^2) + (epsilon_*(rob_diam/2 + obs_diam/2))];
71     end
72 end
73
74 vp_lims = 0.1;
75 vn_lims = 0.1;
76 wp_lims = 0.1;
77 wn_lims = 0.1;
78 for k = 1:N-1
79     g = [g ; (U(1,k)-U(1,k+1))];
80 end
81
82 for k = 1:N-1
83     g = [g ; (U(2,k)-U(2,k+1))];
84 end
85
86 %
87 % make the decision variable one column vector
88 OPT_variables = [reshape(X,3*(N+1),1);reshape(U,2*N,1)];
89
90 nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
91
92 opts = struct;
93 opts.ipopt.max_iter = 10000;
94 opts.ipopt.print_level =0;%0,3
95 opts.print_time = 0;
96 opts.ipopt.acceptable_tol =1e-8;
97 opts.ipopt.acceptable_obj_change_tol = 1e-6;
98
99 solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
100
101 args = struct;
102 % equality constraints
103 args.lbg(1:3*(N+1)) = 0;
104 args.ubg(1:3*(N+1)) = 0;
```

```
105
106 % obstacle constraints
107 length_obs_const = 3*(N+1)+length(obs_x)*(N+1);
108 args.lbg(3*(N+1)+1 : length_obs_const) = -inf;
109 args.ubg(3*(N+1)+1 : length_obs_const) = 0;
110
111 % linear velocity decomposition constraints
112 args.lbg(length_obs_const+1 : length_obs_const+(N-1)) = -vn_lims;
113 args.ubg(length_obs_const+1 : length_obs_const+(N-1)) = vp_lims;
114
115 length_vp = length_obs_const+(N-1);
116 % angular velocity decomposition constraints
117 args.lbg(length_vp+1 : length_vp+(N-1)) = -wn_lims;
118 args.ubg(length_vp+1 : length_vp+(N-1)) = wp_lims;
119
120 % Bounds on the state variables (Boundary Conditions/Outer walls)
121 args.lbx(1:3:3*(N+1),1) = -4.2; %state x lower bound
122 args.ubx(1:3:3*(N+1),1) = 4.2; %state x upper bound
123 args.lbx(2:3:3*(N+1),1) = -4.2; %state y lower bound
124 args.ubx(2:3:3*(N+1),1) = 4.2; %state y upper bound
125 args.lbx(3:3:3*(N+1),1) = -inf; %state theta lower bound
126 args.ubx(3:3:3*(N+1),1) = inf; %state theta upper bound
127
128 % Bounds on the control Variables (Linear and angular velocity)
129 args.lbx(3*(N+1)+1:2:3*(N+1)+2*N,1) = -0.4; %v lower bound
130 args.ubx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_max; %v upper bound
131 args.lbx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_min; %omega lower bound
132 args.ubx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_max; %omega upper bound
133
134 %
135 prompt = {'Enter␣x-Value:','Enter␣y-Value:', 'Enter␣theta-Value␣(Rad)'};
136 dlgtitle = 'Initial␣estimate';
137 dims = [1 35];
138 definput = {'-4.0','-4.0', '0'};
139 answer = inputdlg(prompt,dlgtitle,dims,definput);
140
141 x_initial = str2double(answer{1,1});
142 y_initial = str2double(answer{2,1});
143 theta_initial = str2double(answer{3,1});
144
145 % Initial condition
146 x0 = [x_initial ; y_initial ; theta_initial];
```

```
147
148 prompt = {'Enter␣x-Value:','Enter␣y-Value:', 'Enter␣theta-Value␣(Rad)'};
149 dlgtitle = 'Goal␣Position';
150 dims = [1 35];
151 definput = {'4','4', '0'};
152 answer = inputdlg(prompt,dlgtitle,dims,definput);
153
154 x_final = str2double(answer{1,1});
155 y_final = str2double(answer{2,1});
156 theta_final = str2double(answer{3,1});
157
158 % Reference posture.
159 xs = [x_final ; y_final ; theta_final]; % Reference posture.
160
161 t0 = 0;
162 t(1) = t0;
163 u0 = zeros(N,2);        % two control inputs for each robot
164 X0 = repmat(x0,1,N+1)'; % initialization of the states decision variables
165
166 sim_tim = 30; % Maximum simulation time
167
168 % Start MPC
169 mpciter = 0;
170 %
171 main_loop = tic;
172 while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)
173     args.p   = [x0;xs]; % set the values of the parameters vector
174     % initial value of the optimization variables
175     args.x0  = [reshape(X0',3*(N+1),1);reshape(u0',2*N,1)];
176     % Run the NLP solver
177     sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
178         'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
179
180     % get controls for N+1 only
181     u = reshape(full(sol.x(3*(N+1)+1:end))',2,N)';
182     t(mpciter+1) = t0;
183
184     % Apply the control and shift the solution
185     [t0, x0, u0] = shift(T, t0, x0, u,f);
186
187     % Shift trajectory to initialize the next step
188     X0 = reshape(full(sol.x(1:3*(N+1)))',3,N+1)';
```

```
189     X0 = [X0(2:end,:);X0(end,:)];
190
191     mpciter = mpciter + 1;
192 end
193 main_loop_time = toc(main_loop);
194 ss_error = norm((x0-xs),2)
195 average_mpc_time = main_loop_time/(mpciter+1)
```

Annex B.2. *Matlab script for MPC obstacle avoidance adapted from [43]*

B.3 Matlab MPC code - MiR Shared Control

```
1  clear all
2  close all
3  clc
4
5  %
6  % CasADi v3.4.5
7  addpath('/home/devalla/MATLAB_CasADi_test/casadi_matlab')
8  import casadi.*
9
10
11 %
12 try
13     rosinitd
14 catch ME
15     fprintf('ROS Node is already initialised');
16 end
17
18 velocity_pub = rospublisher('/cmd_vel', 'geometry_msgs/Twist');
19 robot_vel = rosmessage(velocity_pub);
20
21 pub_initial = rospublisher('/initial_pose', 'geometry_msgs/Pose');
22 initial_pose = rosmessage(pub_initial);
23
24 pred_path = rospublisher('predicted_path', 'nav_msgs/Path');
25 predicted_path = rosmessage(pred_path);
26 predicted_path.Header.FrameId = "/map";
27 point_pose = rosmessage('geometry_msgs/PoseStamped');
28
29 current_state_sub = rossubscriber('/gazebo/model_states',...
30     'gazebo_msgs/ModelStates');
31
```

```
32 if_quit_loop = rossubscriber('/end_sim', 'std_msgs/Bool');
33 goal_pose = rossubscriber('/leader_point', 'geometry_msgs/PoseStamped');
34 pause(1);
35
36 %
37 N =15; % prediction horizon
38 rob_diam = 0.9;
39
40 v_max = 0.8; v_min = -v_max;
41 omega_max = 0.8; omega_min = -omega_max;
42
43 x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
44 states = [x;y;theta]; n_states = length(states);
45
46 v = SX.sym('v'); omega = SX.sym('omega');
47 controls = [v;omega]; n_controls = length(controls);
48
49 % Kinematic System Model r.h.s
50 rhs = [v*cos(theta);v*sin(theta);omega];
51
52 f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
53 % Decision variables (controls)
54 U = SX.sym('U',n_controls,N);
55 % Decision Variables (states)
56 X = SX.sym('X',n_states,(N+1));
57
58 % parameters (which include at the initial state of the robot and the reference
   state)
59 P = SX.sym('P',n_states + n_states);
60
61 obj = 0; % Objective function
62 g = [];  % constraints vector
63
64 % Weighting Matrices
65 Q = zeros(3,3); Q(1,1) = 6;Q(2,2) = 6;Q(3,3) = 4;
66 R = zeros(2,2); R(1,1) = 0.3; R(2,2) = 0.15;
67
68 % Symbolically compute constraints and Objective Function
69 st  = X(:,1); % initial state
70 g = [g;st-P(1:3)]; % initial condition constraints
71
72 T = 0.25; % dt in Seconds
```

```
% Define the equality constraints as symbolics
for k = 1:N
    st = X(:,k);  con = U(:,k);
    obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
    st_next = X(:,k+1);
    f_value = f(st,con);
    st_next_euler = st+ (T*f_value);
    g = [g;st_next-st_next_euler]; % compute constraints
end
% Add constraints for collision avoidance (Inequality contraint)

obs_x = [-4 -2 0.5 3 -2.5 0 2 -4 -1 3 -2 1 4 -1 1 3]; % meters
obs_y = [3.5 3 3.5 4 1.5 1.5 2 0 0 0.5 -2 -1 -1 -4 -3 -3]; % meters

obs_diam = 0.45; % meters

for k = 1:N+1
    for num_obs = 1:length(obs_x)
        g = [g ; -sqrt((X(1,k)-obs_x(num_obs))^2+(X(2,k)-...
            obs_y(num_obs))^2) + (rob_diam/2 + obs_diam/2)];
    end
end

vp_lims = 0.1;
vn_lims = 0.1;
wp_lims = 0.1;
wn_lims = 0.1;
for k = 1:N-1
    g = [g ; (U(1,k)-U(1,k+1))];
end

for k = 1:N-1
    g = [g ; (U(2,k)-U(2,k+1))];
end

%
% make the decision variable one column vector
OPT_variables = [reshape(X,3*(N+1),1);reshape(U,2*N,1)];

nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);

opts = struct;
```

```
115 opts.ipopt.max_iter = 10000;
116 opts.ipopt.print_level =0;%0,3
117 opts.print_time = 0;
118 opts.ipopt.acceptable_tol =1e-8;
119 opts.ipopt.acceptable_obj_change_tol = 1e-6;
120 solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
121
122 args = struct;
123 % equality constraints
124 args.lbg(1:3*(N+1)) = 0;
125 args.ubg(1:3*(N+1)) = 0;
126
127 % obstacle constraints
128 length_obs_const = 3*(N+1)+length(obs_x)*(N+1);
129 args.lbg(3*(N+1)+1 : length_obs_const) = -inf;
130 args.ubg(3*(N+1)+1 : length_obs_const) = 0;
131
132 % linear velocity decomposition constraints
133 args.lbg(length_obs_const+1 : length_obs_const+(N-1)) = -vn_lims;
134 args.ubg(length_obs_const+1 : length_obs_const+(N-1)) = vp_lims;
135
136 length_vp = length_obs_const+(N-1);
137 % angular velocity decomposition constraints
138 args.lbg(length_vp+1 : length_vp+(N-1)) = -wn_lims;
139 args.ubg(length_vp+1 : length_vp+(N-1)) = wp_lims;
140
141 % Bounds on the state variables (Boundary Conditions/Outer walls)
142 args.lbx(1:3:3*(N+1),1) = -4.2; %state x lower bound
143 args.ubx(1:3:3*(N+1),1) = 4.2; %state x upper bound
144 args.lbx(2:3:3*(N+1),1) = -4.2; %state y lower bound
145 args.ubx(2:3:3*(N+1),1) = 4.2; %state y upper bound
146 args.lbx(3:3:3*(N+1),1) = -inf; %state theta lower bound
147 args.ubx(3:3:3*(N+1),1) = inf; %state theta upper bound
148
149 % Bounds on the control Variables (Linear and angular velocity)
150 args.lbx(3*(N+1)+1:2:3*(N+1)+2*N,1) = -0.4; %v lower bound
151 args.ubx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_max; %v upper bound
152 args.lbx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_min; %omega lower bound
153 args.ubx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_max; %omega upper bound
154
155 %
156 prompt = {'Enter␣x-Value:','Enter␣y-Value:', 'Enter␣theta-Value␣(Rad)'};
```

```
157 dlgtitle = 'Initial estimate';
158 dims = [1 35];
159 definput = {'-4.0','-4.0', '0.0'};
160 answer = inputdlg(prompt,dlgtitle,dims,definput);
161
162 x_initial = str2double(answer{1,1});
163 y_initial = str2double(answer{2,1});
164 theta_initial = str2double(answer{3,1});
165 % Initial condition
166 x0 = [x_initial ; y_initial ; theta_initial];
167
168 eul = [theta_initial 0 0];
169 quat = eul2quat(eul);
170 unpuase_gazebo = call(rossvcclient('/gazebo/unpause_physics'));
171
172 % Set robot to desired start pose on gazebo
173 set_pose_gazebo = rossvcclient('/gazebo/set_model_state');
174 set_pose_msg = rosmessage(set_pose_gazebo);
175 set_pose_msg.ModelState.ModelName = 'mir';
176 set_pose_msg.ModelState.Pose.Position.X = x_initial;
177 set_pose_msg.ModelState.Pose.Position.Y = y_initial;
178 set_pose_msg.ModelState.Pose.Position.Z = 0;
179 set_pose_msg.ModelState.Pose.Orientation.W = quat(1);
180 set_pose_msg.ModelState.Pose.Orientation.X = quat(2);
181 set_pose_msg.ModelState.Pose.Orientation.Y = quat(3);
182 set_pose_msg.ModelState.Pose.Orientation.Z = quat(4);
183
184 initial_pose.Position.X = x_initial;
185 initial_pose.Position.Y = y_initial;
186 initial_pose.Orientation.W = quat(1);
187 initial_pose.Orientation.X = quat(2);
188 initial_pose.Orientation.Y = quat(3);
189 initial_pose.Orientation.Z = quat(4);
190
191 set_position_gazebo = call(set_pose_gazebo, set_pose_msg);
192 send(pub_initial, initial_pose);
193
194 final_pose = receive(goal_pose, 3);
195 x_final = final_pose.Pose.Position.X;
196 y_final = final_pose.Pose.Position.Y;
197 theta_quat_w = final_pose.Pose.Orientation.W;
198 theta_quat_x = final_pose.Pose.Orientation.X;
```

```
199 theta_quat_y = final_pose.Pose.Orientation.Y;
200 theta_quat_z = final_pose.Pose.Orientation.Z;
201 theta_eul = quat2eul([theta_quat_w, theta_quat_x, theta_quat_y, theta_quat_z]);
202
203 theta_final = theta_eul(1);
204
205 % Reference posture.
206 xs = [x_final ; y_final ; theta_final]; % Reference posture.
207
208 t0 = 0;
209 t(1) = t0;
210 u0 = zeros(N,2);        % two control inputs for each robot
211 X0 = repmat(x0,1,N+1)'; % initialization of the states decision variables
212
213 % Start MPC
214 mpciter = 0;
215 quit_while = receive(if_quit_loop, 10);
216
217 main_loop = tic;
218 while (quit_while.Data == 0)
219     args.p   = [x0;xs]; % set the values of the parameters vector
220     % initial value of the optimization variables
221     args.x0  = [reshape(X0',3*(N+1),1);reshape(u0',2*N,1)];
222
223 % Run the NLP Solver
224     sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
225         'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
226
227 % get controls only for N+1
228     u = reshape(full(sol.x(3*(N+1)+1:end))',2,N)';
229     t(mpciter+1) = t0;
230
231     % Send the control values to the MiR Gazebo Simulation
232     robot_vel.Linear.X = u(1,1);
233     robot_vel.Angular.Z = u(1,2);
234     send(velocity_pub, robot_vel);
235     send(pred_path, predicted_path);
236
237     % Recieve current robot position from Gazebo
238     rob_state = receive(current_state_sub,1);
239
240     isMir = cellfun(@(x)isequal(x,'mir'),rob_state.Name);
```

```
241     [row,col] = find(isMir);
242     robot_pose_x = rob_state.Pose(row,1).Position.X;
243     robot_pose_y = rob_state.Pose(row,1).Position.Y;
244
245     eulers = quat2eul([rob_state.Pose(row,1).Orientation.W,...
246         rob_state.Pose(row,1).Orientation.X, ...
247         rob_state.Pose(row,1).Orientation.Y,...
248         rob_state.Pose(row,1).Orientation.Z]);
249
250     robot_orientation = eulers(1);
251
252     % propogate values for next prediction
253     x0 = [robot_pose_x; robot_pose_y; robot_orientation];
254     t0 = t0 + T;
255     u0 = [u(2:size(u,1),:);u(size(u,1),:)];
256     X0 = reshape(full(sol.x(1:3*(N+1)))',3,N+1)'; %
257
258     % Shift trajectory to initialize the next step
259     X0 = [X0(2:end,:);X0(end,:)];
260     final_pose = receive(goal_pose, 3);
261     x_final = final_pose.Pose.Position.X;
262     y_final = final_pose.Pose.Position.Y;
263     theta_quat_w = final_pose.Pose.Orientation.W;
264     theta_quat_x = final_pose.Pose.Orientation.X;
265     theta_quat_y = final_pose.Pose.Orientation.Y;
266     theta_quat_z = final_pose.Pose.Orientation.Z;
267     theta_eul = quat2eul([theta_quat_w, theta_quat_x, ...
268         theta_quat_y, theta_quat_z]);
269     theta_final = theta_eul(1);
270
271     xs = [x_final ; y_final ; theta_final];
272
273     mpciter = mpciter + 1;
274     quit_while = receive(if_quit_loop,1);
275 end
276 main_loop_time = toc(main_loop);
277 ss_error = norm((x0-xs),2)
278 average_mpc_time = main_loop_time/(mpciter+1)
```

Annex B.3. *Matlab script for MPC based Shared Control through ROS.*

B.4 Code for human joystick inputs

```
1  # This code is an extract from the Jpystick Control
2  # node from Andrew Dai
3
4  #!/usr/bin/env python
5  import rospy
6  from geometry_msgs.msg import Twist
7  from sensor_msgs.msg import Joy
8  from std_msgs.msg import Bool
9
10
11
12 def callback(data):
13     twist = Twist()
14     if_ending = Bool()
15     if (data.buttons[0] == 1 and data.buttons[1] == 0):
16         if_ending = False
17         twist.linear.x = 0.9*data.axes[1]
18         twist.angular.z = 0.9*data.axes[0]
19
20     elif (data.buttons[0] == 0 and data.buttons[1] == 0):
21         if_ending = False
22         twist.linear.x = 0
23         twist.angular.z = 0
24
25     elif (data.buttons[1] == 1):
26         if_ending = True
27         twist.linear.x = 0
28         twist.angular.z = 0
29
30     pub_vel.publish(twist)
31     pub_termin.publish(if_ending)
32
33 # Intializes everything
34 def start():
35     global pub_vel
36     global pub_termin
37     pub_vel = rospy.Publisher('/cmd_vel_leader', Twist,
38                               queue_size = 10)
39     pub_termin = rospy.Publisher('/end_sim', Bool,
40                               queue_size = 2)
41     # subscribed to joystick inputs on topic "joy"
42     rospy.Subscriber("joy", Joy, callback)
```

```
43      # starts the node
44      rospy.spin()
45
46  if __name__ == '__main__':
47      rospy.init_node('Mir_Tele')
48      start()
```

Annex B.4. *Script for subscribing joystick commands and publishing as twist commands.*

B.5 Code for updating pose of user controlled point

```
1  #!/usr/bin/env python
2  import rospy
3  from geometry_msgs.msg import PoseStamped, Twist, Pose
4  from gazebo_msgs.msg import ModelState
5  from gazebo_msgs.srv import SetModelState
6  from visualization_msgs.msg import Marker
7  import numpy as np
8  from tf.transformations import quaternion_from_euler
9  from tf.transformations import euler_from_quaternion
10
11
12 global X
13 global Y
14 global THETA
15 X = 0
16 Y = 0
17 THETA = 0
18
19 def angleWrapping(th):
20     while th < -np.pi:
21         th += 2 * np.pi
22     while th > np.pi:
23         th -= 2 * np.pi
24     return th
25
26
27 def pose_clbk(data):
28     global X
29     global Y
30     global THETA
31     X = data.position.x
32     Y = data.position.y
```

```
33      THETA = euler_from_quaternion([data.orientation.x,
34                                     data.orientation.y,
35                                     data.orientation.z,
36                                     data.orientation.w])[2]
37
38
39  def callback(data):
40      global X
41      global Y
42      global THETA
43      linear_vel_v = data.linear.x
44      linear_vel_w = data.angular.z
45      leader_gazebo = ModelState()
46      leader_gazebo.model_name = 'leader'
47      leader_pose.header.stamp = rospy.Time.now()
48      leader_marker.header.stamp = rospy.Time.now()
49
50      dt = 0.01
51
52      X = X + (linear_vel_v * np.cos(THETA) * dt)
53      Y = Y + (linear_vel_v * np.sin(THETA) * dt)
54      THETA = THETA + linear_vel_w * dt
55      THETA = angleWrapping(THETA)
56
57      quat = quaternion_from_euler(0,0,THETA)
58
59      leader_pose.pose.position.x = X
60      leader_pose.pose.position.y = Y
61      leader_pose.pose.orientation.x = quat[0]
62      leader_pose.pose.orientation.y = quat[1]
63      leader_pose.pose.orientation.z = quat[2]
64      leader_pose.pose.orientation.w = quat[3]
65
66      leader_gazebo.pose.position.x = X
67      leader_gazebo.pose.position.y = Y
68      leader_gazebo.pose.orientation.x = quat[0]
69      leader_gazebo.pose.orientation.y = quat[1]
70      leader_gazebo.pose.orientation.z = quat[2]
71      leader_gazebo.pose.orientation.w = quat[3]
72
73      leader_marker.pose.position.x = X
74      leader_marker.pose.position.y = Y
```

```
75
76      try:
77          set_state = rospy.ServiceProxy('/gazebo/set_model_state',
78                                         SetModelState)
79          resp = set_state( leader_gazebo )
80
81      except rospy.ServiceException, e:
82          print "Service call failed: %s" % e
83
84      pub_circle.publish(leader_marker)
85      pub_square.publish(final_marker)
86      pub_point.publish(leader_pose)
87
88
89  leader_pose = PoseStamped()
90  leader_pose.header.frame_id = "/map"
91
92  leader_marker = Marker()
93  leader_marker.header.frame_id = "/map"
94  leader_marker.ns = "cicle_for_leader"
95  leader_marker.id = 0
96  leader_marker.type = leader_marker.CYLINDER
97  leader_marker.action = 0
98  leader_marker.pose.orientation.x = 0
99  leader_marker.pose.orientation.y = 0
100 leader_marker.pose.orientation.z = 0
101 leader_marker.pose.orientation.w = 1
102 leader_marker.scale.x = 0.88
103 leader_marker.scale.y = 0.88
104 leader_marker.scale.z = 0.01
105 leader_marker.color.r = 0.0
106 leader_marker.color.g = 1.0
107 leader_marker.color.b = 0.0
108 leader_marker.color.a = 1.0
109
110 final_marker = Marker()
111 final_marker.header.frame_id = "/map"
112 final_marker.ns = "square_final_area"
113 final_marker.id = 0
114 final_marker.type = leader_marker.CUBE
115 final_marker.action = 0
116 final_marker.pose.position.x = 4.0
```

```
117 final_marker.pose.position.y = 2.5
118 final_marker.pose.orientation.x = 0
119 final_marker.pose.orientation.y = 0
120 final_marker.pose.orientation.z = 0
121 final_marker.pose.orientation.w = 1
122 final_marker.scale.x = 2
123 final_marker.scale.y = 2
124 final_marker.scale.z = 0.01
125 final_marker.color.r = 0.0
126 final_marker.color.g = 1.0
127 final_marker.color.b = 1.0
128 final_marker.color.a = 0.5
129
130
131 # Intializes everything
132 def start():
133     global pub_circle
134     global pub_point
135     global pub_square
136
137     pub_point = rospy.Publisher('/leader_point', PoseStamped,
138                                 queue_size = 3)
139     pub_circle = rospy.Publisher('circle_leader', Marker,
140                                 queue_size = 0)
141     pub_square = rospy.Publisher('final_region', Marker,
142                                 queue_size = 0)
143     # subscribed to joystick inputs on topic "joy"
144     rospy.Subscriber('/cmd_vel_leader', Twist, callback)
145     rospy.Subscriber('/initial_pose', Pose, pose_clbk)
146
147     # starts the node
148     rospy.spin()
149
150 if __name__ == '__main__':
151     rospy.init_node('Leader_move')
152     start()
```

Annex B.5. *Script for subscribing Twist commands from the joystick node and publishing the updated pose of human controlled point.*

B.6 Code for direct tele/operation of the robot

```
1 # This code is an extract from the Joystick control
```

```
2  # node from Andrew Dai
3
4  #!/usr/bin/env python
5  import rospy
6  from geometry_msgs.msg import Twist
7  from sensor_msgs.msg import Joy
8  from std_msgs.msg import Bool
9
10
11
12 def callback(data):
13     twist = Twist()
14
15     if_moving = Bool()
16     if (data.buttons[0] == 1):
17         twist.linear.x = 0.8*data.axes[1]
18         twist.angular.z = 0.8*data.axes[0]
19         if_moving = True
20
21     elif (data.buttons[0] == 0):
22         twist.linear.x = 0
23         twist.angular.z = 0
24         if_moving = False
25
26
27     pub_vel.publish(twist)
28     pub_moving.publish(if_moving)
29
30 # Intializes everything
31 def start():
32     global pub_vel
33     global pub_moving
34     pub_vel = rospy.Publisher('/cmd_vel', Twist,
35                               queue_size = 10)
36     pub_moving = rospy.Publisher('/robot_moving', Bool,
37                                  queue_size = 2)
38     rospy.Subscriber("joy", Joy, callback)
39     # starts the node
40     rospy.spin()
41
42 if __name__ == '__main__':
43     rospy.init_node('Mir_Tele')
```

```
44          start()
```

Annex B.6. *Ros python script used for direct tele-operation of the MiR in the simulation environment.*

C USER TESTS OF SHARED CONTROL IMPLEMENTATION

C.1 Consent Form

User testing of the Shared Control Implementation

Informed Consent Form

This is a user-based testing performed towards the evaluation of the shared control approach implemented as a part of my master's thesis on the topic "Shared Control for Mobile Robots".

Tasks during the study

In this session you will have two major tasks to perform. These user tests will help in attaining an unbiased and neutral understanding on the performance of the system. In both the task contexts you will perform the tasks thrice, once by using manual teleoperation with zero autonomy and the other two times using the shared control algorithm once without any training to use the implemented approach and once with some training. As part of the user testing you will perform the following tasks:

1. **How safe can you manoeuvre?**
   In this task you will navigate the robot in a 10X10 metre environment for a fixed time period of 2 minutes. As a safety evaluation, the number of collisions occurring will be recorded.
2. **How quick can you manoeuvre?**
   In the second task, you will navigate the robot from a starting point, towards a final designated region highlighted on the floor of the environment. During this test, the total time taken for the task completion will be recorded.

Confidentiality of the user test.

No pictures or videos of the participants will be recorded as a part of the user testing for the shared control system. The tests performed are for testing the performance of the system and not the user. The results from this session and other sessions will be published as a part of thesis. The findings will be published anonymously with no inclusion of names in the thesis.

Consent

I have read the description of the user study and my rights as a participant. I agree to voluntarily participate in the user testing.

Name_____

Signature and Date _____

C.2 User Test Data

Test Number	Collisions occurred during Task 1		Time taken for comletion of Task 2		
	Direct Tele-operation	Tele-operation v	Direct Tele-operation	Tele-operation via Shared Control - No training	Tele-operation via Shared Control - With training
Test 1	3	0	16.173	19.542	17.719
Test 2	10	0	19.518	19.485	19.392
Test 3	2	0	16.33	19.503	17.632
Test 4	4	0	20.84	19	
Test 5	3	0	17.265	20.355	16.241
Test 6	3	0	19.976	18.082	19.034
Test 7	4	0	17.13	19.704	18.61
Total	29	0	127.232	135.671	108.628
Average	4.142857143	0	18.176	19.38157143	18.10466667