Unmanned Ground vehicles: Adaptive Control System for Real-Time Rollover Prevention

by

MALAVI CLIFFORD MLATI

Submitted in accordance with the requirements

for the degree of

MAGISTER TECHNOLOGIAE: ELECTRICAL ENGINEERING

at the

University of South Africa

Supervisor: Prof. Zenghui Wang

October 2019

Declaration

I declare that this research is my own, unaided work. It has been submitted for the Magister Technologiae in Electrical Engineering in the Department of Electrical and Mining Engineering at UNISA. It has not been submitted before for any degree, diploma or other examination at any other tertiary educational institution.

[illegible]
……………………………………..

MALAVI CLIFFORD MLATI

October 2019

This dissertation is dedicated to my Daughter Ntsako Rhulani Keisha Mlati and Son Ntsembo Xiviko Malavi Mlati

Acknowledgement

I will like to express my gratitude to Professor Zenghui Wang, Department of Electrical and Mining Engineering for his support and willingness to review this dissertation throughout the period of my study.

I will like to thank my wife Tlangelani Mlati for her support and encouragement during this research.

Finally, I want to thank my Sister, Brother and Parents (Sarah &David Mlati) for their support throughout the years of my studies.

Abstract

Real-Time Rollover prevention of Unmanned Ground Vehicle (UGV) is very paramount to its reliability and survivability mostly when operating on unknown and rough terrains like mines or other planets.Therefore this research presents the method of real-time rollover prevention of UGVs making use of Adaptive control techniques based on Recursive least Squares (RLS) estimation of unknown parameters, in order to enable the UGVs to adapt to unknown hush terrains thereby increasing their reliability and survivability.

The adaptation is achieved by using indirect adaptive control technique where the controller parameters are computed in real time based on the online estimation of the plant's (UGV) parameters (Rollover index and Roll Angle) and desired UGV's performance in order to appropriately adjust the UGV speed and suspension actuators to counter-act the vehicle rollover.

A great challenge of indirect adaptive control system is online parameter identification, where in this case the RLS based estimator is used to estimate the vehicles rollover index and Roll Angle from lateral acceleration measurements and height of the centre of gravity of the UGV. RLS is suitable for online parameter identification due to its nature of updating parameter estimate at each sample time.

The performance of the adaptive control algorithms and techniques is evaluated using Matlab Simulink® system model with the UGV Model built using SimMechanics physical modelling platform and the whole system runs within Simulink environment to emulate real world application.

The simulation results of the proposed adaptive control algorithm based on RLS estimation, show that the adaptive control algorithm does prevent or minimize the likely hood of vehicle rollover in real time

Keywords – Adaptive Control System; Unmanned Ground Vehicle, Roll Angle; Rollover Index; Recursive Least Squares; lateral acceleration; Parameter Estimation.

Table of Contents

List of Figures

Glossary of Abriviations

a_y – Lateral Acceleration

DC – Direct Current

ESA – European Space Agency

g – Gravetitional acceleration

L – Wheel base

l_W - Track with

LMS – Least Mean Square

m – Vehicle mass

NASA – National Aeronautics and Space Administration

PID – Proportional Integral Derivative

PWM – Pulse With Modulation

RPM – Revelotion Per Minutes

r – radius of the vehicle path at a given steering angle

RI – Rollover index

RLS – Recursive least square

UGV – Unmmanned Ground Vehicle

v_x – Vehicles logitudinal speed

δ – Steering angle

Chapter 1 Introduction

1.1 Overview

Unmanned Ground vehicles (UGVs) are the vehicles which are operated on the ground without on board human presence but remotely controlled. They are used mostly where it's impossible or dangerous for human presence (e.g. rescue missions (mines, military), scientific exploration) an example of such vehicles are the NASA's planet mars exploration vehicle named MER-A spirit and MER-B opportunity [1, 2], Fig 1.1 shows an example of mars two exploration rovers.

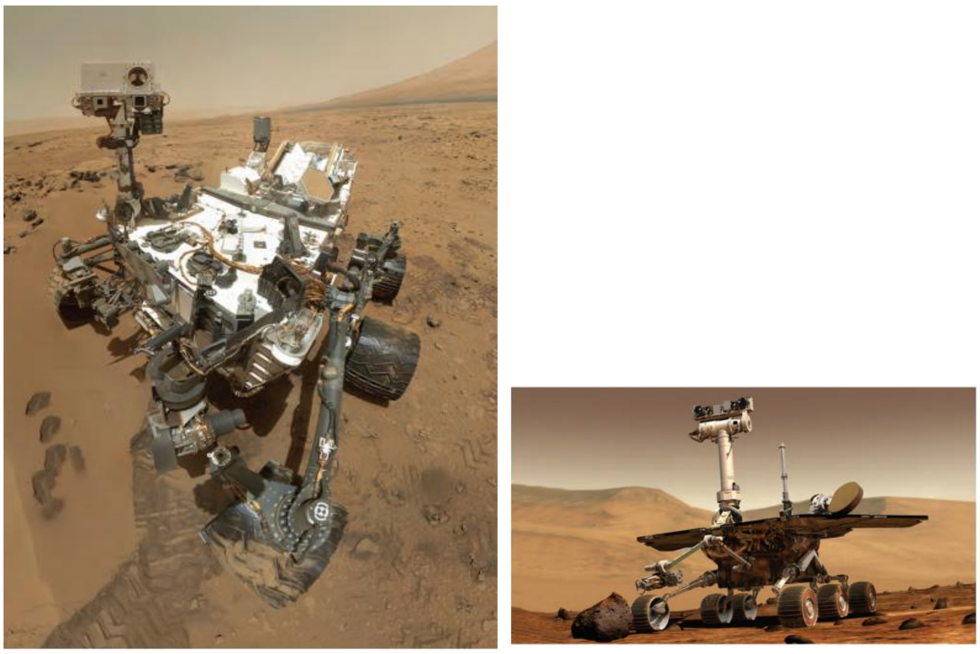


Fig. 1.1 Mars Exploration Rovers [1]

As manned vehicles the unmanned ground vehicles are also prone to rollover moreover when negotiating unknown terrain. One of the root causes of UGV failures is human errors which are mistakes or slips which can lead to the UGV rollover [3], for example the operator drives the UGV unwittingly on a steep slope which induces rolling moment on the vehicle or makes a sharp turn at high speed which induces high lateral forces.

The study focuses on the use of adaptive control techniques to prevent vehicle rollover in real time , where adaptive control is defined as the control method used by a controller, which must adapt to a system with parameters which vary, or are initially uncertain, it provides techniques for the automatic adjustment of control parameters in real time either to achieve or to maintain a desired level of control system performance when the dynamic parameters of the process to be controlled are unknown and/or time-varying [4].

1.2 Background

There is number of Unmanned Ground vehicles (UGVs) developed around the world, dating back to the 1930th with many potential application and the demand for them is ever increasing ranging from military, surveillance, transportation, exploration and industrial usage meaning the UGV's reliability and survivability is paramount.

There is considerable number of research done on UGVs or mobile robots hazard avoidance [5, 6] which undoubtedly increases the vehicle reliability and survivability but they don't solve the challenge of uneven terrain which might introduce high lateral vehicle dynamics.

As indicated in [3] improper (human) operation may cause failures in UGVs, with one example in which the operator drove a robot into an area, in the World trade centre site where the incline was too steep for it, because he could not judge the height of the hole.
Fig 1.2 shows the search and rescue robots used in the world Trade centre's rescue operation.

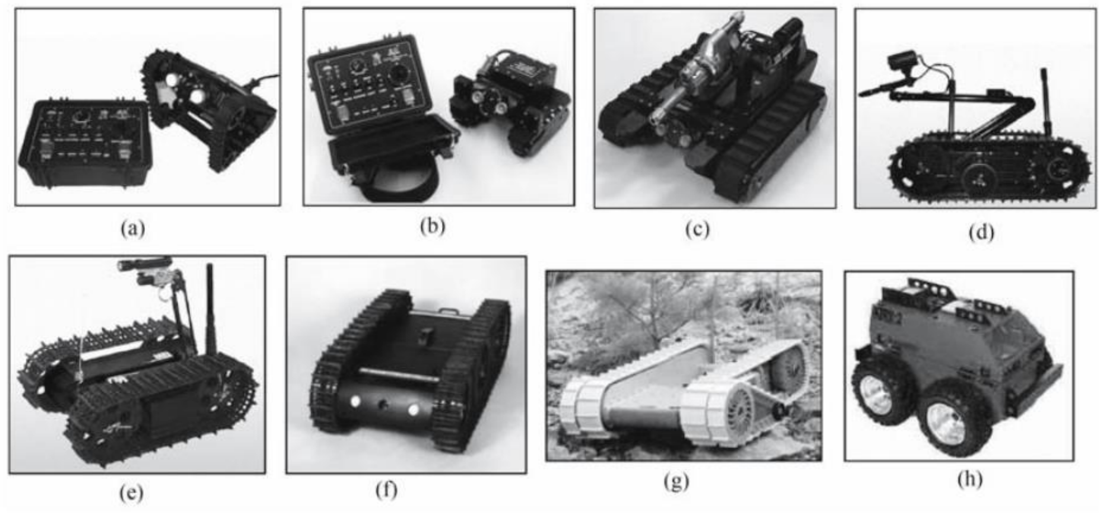


Fig. 1.2 Search and rescue robots used in World Trade Center's rescue operation (a) Micro VGTV; (b) Micro Traces; (c) Mini Traces; (d) Talon; (e) SOLEM; (f) Urbot; (g) Packbot; (h) ATRV [7]

There are two types of vehicle rollover mostly studied on manned vehicle but also applicable to unmanned vehicles: tripped and un-tripped [8] In which un-tripped rollover can be caused by excessive speed when negotiating a curve or during obstacle avoidance or tripped rollover which can be caused by any external disturbance on the vehicle path as illustrated in Fig 1.3 and typically the controller or driver of the UGV won't have any warning in real-time to indicate that the vehicle is on the point of rollover, it all depends on the controllers/drivers experience.

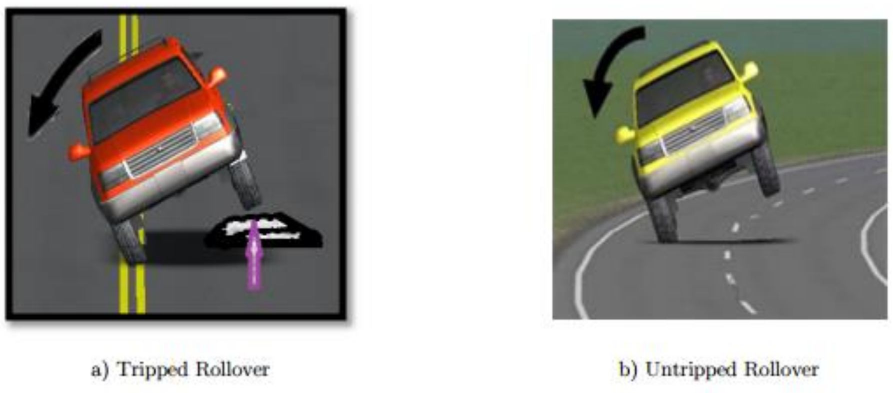


Fig. 1.3 Types of rollovers [9].

1.3 Research Motivation

As manned vehicles unmanned ground vehicles are prone to rollover moreover when negotiating unknown terrain as indicated in [3], one of the root cause of UGV failures is human errors which are mistakes or slips which can lead to the UGV rollover but besides studies and development conducted in relation to vehicle reliability which includes obstacle avoidance [5, 6] they still doesn’t solve the problem with vehicle lateral dynamics which may be induced by this avoidance manoeuvres or by uneven terrain which can lead to vehicle rollover, mostly when the vehicle is operating in unknown terrains like mountains, mines and other planets, therefore a rollover prevention system is paramount to the UGV reliability and survivability.

The main challenge of any rollover prevention system is to determine a cost-effective method of determining a point of rollover in real time, in order to activate appropriate rollover prevention mechanism.

1.4 Research Question

The main questions, which this research is to answer, are:

1. How will Adaptive Control System techniques be cost effectively utilised or applied on a UGV for it to automatically adapt or adjust to environments or terrains which are initially unknown or uncertain in real-time.
2. Which parameter estimation method is suitable for adaptive real time rollover prevention?
3. How will adaptive control techniques and algorithms be utilized for effective and applicable real-time rollover prevention?

1.5 Research Objectives

The aim of the research is to achieve the following:

1. To design a cost-effective adaptive controller for the UGV.
2. To find a suitable estimation method of unknown parameters
3. To investigate the effectiveness and applicability of adaptive control techniques in real-time rollover prevention.

1.6 Delimitation

This study will be limited to the design and evaluation of an adaptive control scheme on UGVs using Matlab Simulink® graphical programming and simulation software.

1.7 Structure of the Dissertation

This section illustrates short description of each chapter contained in the dissertation

Chapter 1: Introduction

This chapter illustrates the overview of the research study and why it has been conducted.

Chapter 2: literature survey

This chapter covers literature review on adaptive control system compared to other control techniques and why an adaptive control technique has been chosen.

It also covers literature on vehicle rollover prevention, anti-collision system and adaptive control parameter estimations.

Chapter 3: Controllers Design

This chapter covers the design of adaptive controller and the formulation of adaptive control algorithms

Chapter 4: Simulation of adaptive controller using Matlab/Simulink

In this chapter the designed adaptive control algorithm for the UGV is simulated using Matlab Simulink to determine its applicability

Chapter 5: Analysis and Discussion of results

This chapter provide the analysis of the simulated results between a UGV without adaptive control system and the one with adaptive control system.

Chapter 6: Conclusion

This chapter gives the overall conclusion on the applicability of adaptive control system in mitigating the UGV’s rollover in real time.

Chapter 2 Literature Survey

2.1 Introduction

Before the research question can be answered a survey or review on previous literature is performed and divided into four sections: Vehicle model, Adaptive Control Systems, Rollover Prevention and Parameter Estimation.

2.2 Vehicle model

For adaptive control system to be properly implemented the vehicle model has to be developed, for the purpose of this dissertation the vehicle model will be based on two crucial parameters: vehicle lateral acceleration and the rollover index.

2.2.1 Lateral acceleration

Fig 2.1 show the Ackerman steering bicycle model of the vehicle where δ is the steering angle of the front wheel, L is the distance between the front axle and rear axle (wheelbase), V_x longitudinal velocity of the vehicle and R is the radius of the vehicle path at a given steering angle.

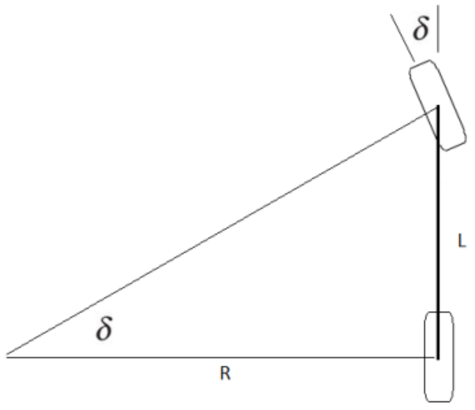


Fig. 2.1 Ackerman steering bicycle model

From the bicycle model in Fig 2.1 lateral acceleration relation with the vehicle velocity can be written as:

$$a_y = \frac{V_x^2}{R} \tag{2.1}$$

With

$$R = \frac{L}{tan\delta} \tag{2.2}$$

2.2.2 Rollover index

Rollover index is defined as the variable that indicates the likelihood for vehicle to roll over [10], and it has to be determined or calculated in real time for appropriate action to be taken to prevent roll over.

Fig. 2.2 shows schematic of a vehicle with a sprung mass that undergoes roll motion which leads to the difference between the vertical tire forces *Fzl* and *Fzr*, which is then used to define the rollover index equation (2.3).

$$RI = \frac{F_{zl}-F_{zr}}{F_{zl}+F_{zr}} \tag{2.3}$$

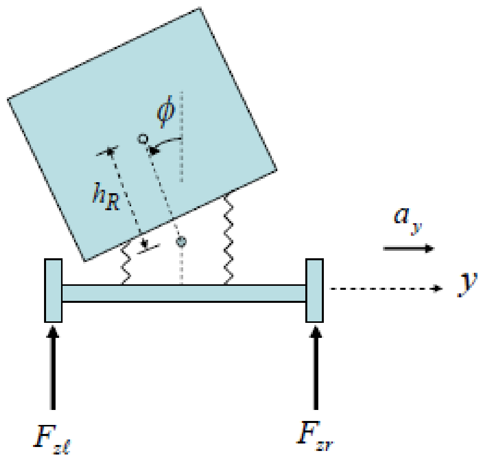


Fig. 2.2 Rollover index using lateral load transfer [10]

Here

RI = rollover index

h_R = height of from the roll centre to the centre of gravity (fixed value) in meters (m)

l_w = track width (fixed value) in meters (m)

a_y = lateral acceleration (measured) in m/s^2

Ø= Roll angle in degrees

But Equation (2.3) is not easy to implement in real-time moreover when the vehicle is in motion as there will be a need to measure vertical tire forces in real time.
Looking at equation (2.3) for one degree of freedom roll dynamics model the total vertical tire forces are given as:

$$F_{zl} + F_{zr} = mg \tag{2.4}$$

And the roll dynamics equation of motion is given as:

$$F_{zl} - F_{zr} = \frac{2ma_{yh_R} \cos \emptyset + 2mgh_R \sin \emptyset}{l_w} \tag{2.5}$$

Therefore:

$$RI = \frac{2h_{c.g} a_y cos\emptyset + 2gh_R sin\emptyset}{l_w g} \tag{2.6}$$

Assuming small roll angles the rollover index is approximated as

$$RI \approx \frac{2h_R a_y}{l_w g} \tag{2.7}$$

But Formula (2.6) is more appropriate for accurate calculation of the rollover index as indicated in [9].

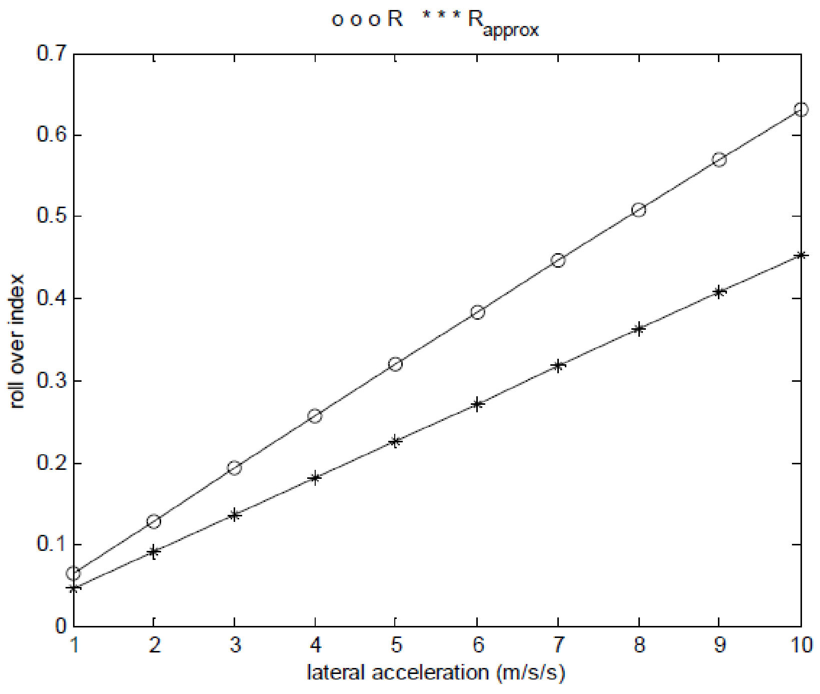


Fig. 2.3 Rollover indices (circle) *R* and (star) *R*approx as a function of lateral Acceleration [8]

"Fig. 2.3 shows the original rollover index R and its approximation R (approx.) as a function of lateral acceleration during steady-state cornering around a circular track for a Volvo XC 90 SUV. It can be seen that the difference between the two curves increases as lateral acceleration increases, resulting in higher error during tight cornering manoeuvres. Furthermore, the error increases with increase in the height of the centre of gravity c.g. Thus, the use of roll angle is important in an accurate calculation of the rollover index. This motivates the need to estimate roll angle.” [8]

2.3 Adaptive Control System

There are number of other control algorithms other than adaptive control which are available or used in the modern automatic control systems e.g. Model Predictive Control and PID Control.

2.3.1 Model Predictive Control

Model Predictive control (MPC) is defined as any algorithm which uses a model of a system to predict its future behaviour and select the most appropriate control action based on an optimality creation.

Fig 2.4 as illustrated in [12] provides the general principle of model predictive control

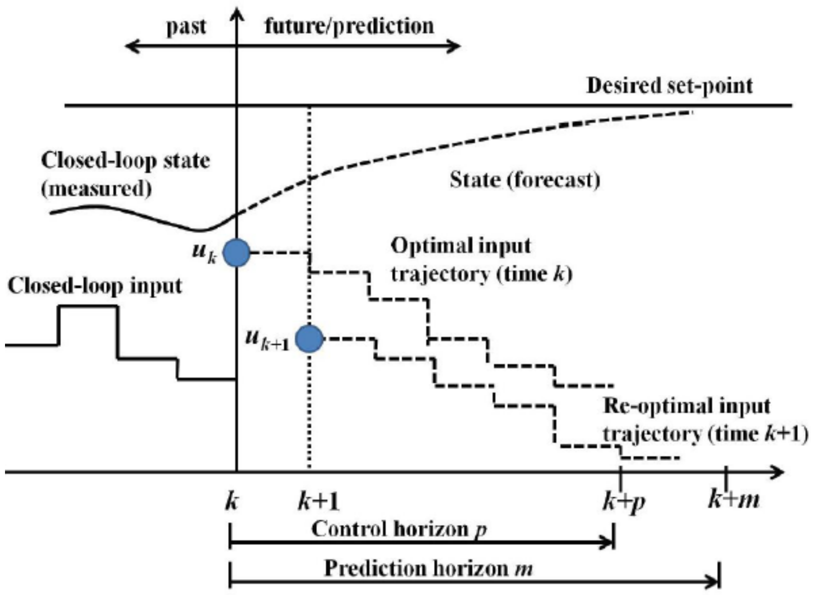


Fig. 2.4 Principle of model predictive control [11]

The MPC consists of reference trajectory which represents the desired target trajectory for the system and the predicted output based on the predicted control input.

However, as per this specific requirements the UGV has to adapt to any environment which is initially unknown, uncertain or time varying, which in this case predictive control needs the past trajectory (inputs and outputs of the system) to make control decisions based upon a comparison between where the plant is and where the model of the plant says it should be, and the model quality plays a vital role in MPC, but in reality model uncertainties always exist [12].

MPC can also be used or has been used on UGVs and ROVERS (space exploration vehicle), e.g. Results of the ESA (European Space Agency) project RobMPC (Robust Model Predictive Control for Space Constraint Systems) could successfully demonstrate that model predictive control (MPC) is definitively applicable for space systems with high dynamics like wheeled vehicles exploring a planetary surface [13].

But besides the accomplishment of the above project any unaccounted dynamics in real world scenarios can results on a model of the system being inaccurate; therefore, the stability and performance of an MPC approach cannot be guaranteed.

2.3.2 PID Control

PID (proportional-integral-derivative) Control is the mostly widely used control strategy in the industry today; it operates by calculating an "error" value as the difference between a measured process variable and a desired set point.

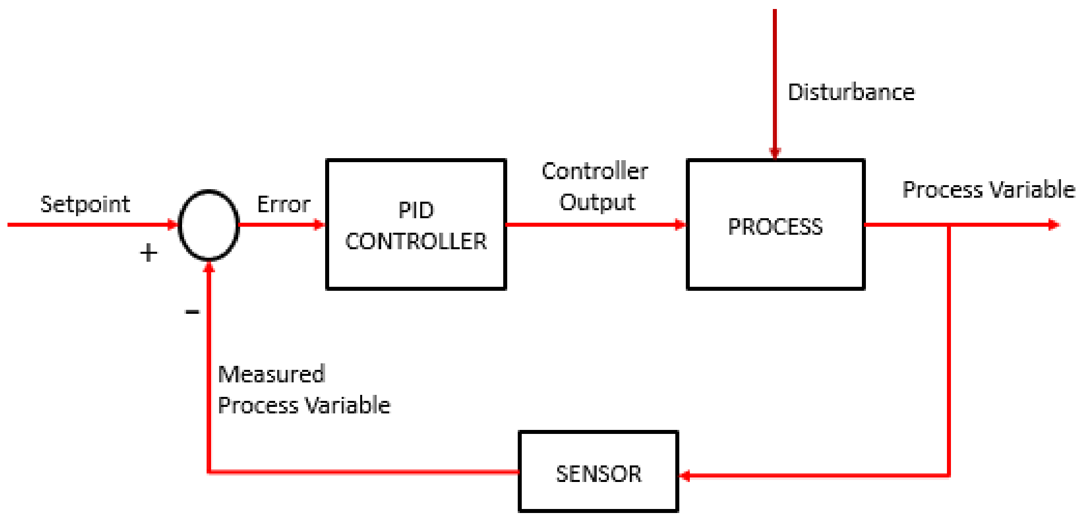


Fig. 2.5 PID controller

Fig 2.5 shows a classical PID control system with the PID controller consisting of three basic coefficients: proportional, integral and derivative which are varied to get optimal response of the system even in the presence of external disturbance on the system process/plant it can always adjust to the desired set-point.

But the classical PID controller parameters cannot adjust or adapt to changes on control systems parameters and it cannot measure/ monitor the performance of the control system in the presence of parameter disturbances.

2.3.3 Adaptive Control

Adaptive control is defined as the control method used by a controller, which must adapt to a system with parameters which vary, or are initially uncertain, it provides techniques for the automatic adjustment of control parameters in real time either to achieve or to maintain a desired level of control system performance when the dynamic parameters of the process to be controlled are unknown and/or time-varying [4].

Classic adaptive control is based on solving plant parameter uncertainty for example the dynamic characteristics of a static inverter depends on the load which can change randomly in time, therefore in order to maintain a desirable control system performance the unknown parameters of the load has to be estimated in order to adjust the inverter's control in real time.

There are number of examples where adaptive control has been demonstrated successfully, one example is illustrated in [14] where the Digital control of uninterrupted power supply has been successfully demonstrated using adaptive control techniques in Fig 2.6; where a Kalman filter is used to estimate the inductor current $\hat{I}_L(k)$and the RLS estimator is used to estimate the unknown parameters $\theta(k)$ as the output voltage was the only measured variable, with the simulation results in Fig 2.7 showing the measured vs the estimated inductor current for a linear load.

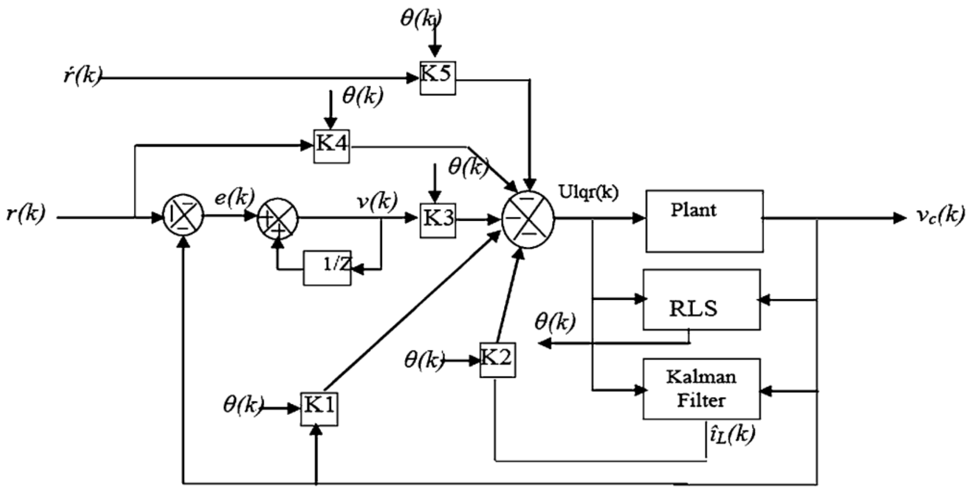


Fig. 2.6 adaptive control system block diagram [14]

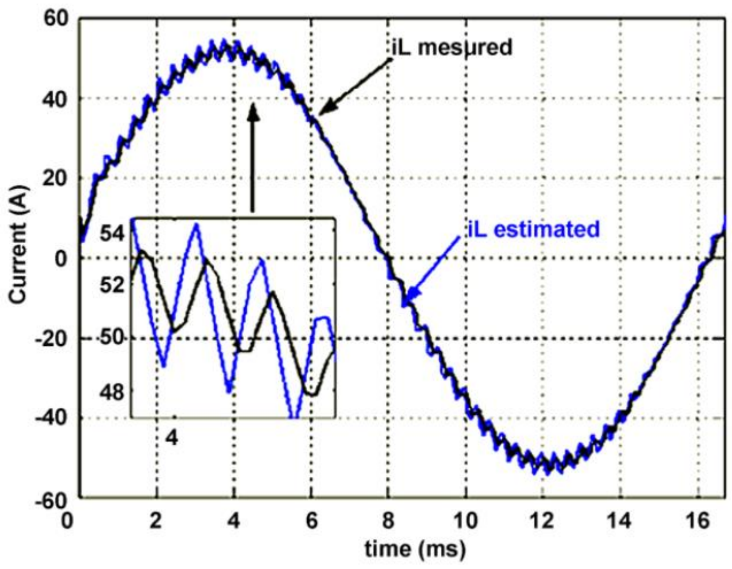


Fig. 2.7 Measured and estimated inductor current for a linear load [14]

For this dissertation the study on adaptive control system is limited to indirect adaptive control scheme.

In an Indirect Adaptive Control Scheme, the adaptation of the controller is done in two stages: (1) on-line estimation of the plant parameters and (2) online computation of the controller parameters based on the current estimated plant model.

This adaptive control scheme will serve the main objective of the UGV requirement/objective which says it should adapt to environments which are initially unknown or time-varying in real-time; as the basic idea of indirect adaptive control is to estimate the plant model on-line (real-time) from input and output measurements and a suitable controller is designed in real-time.

Fig 2.8 is a basic example of an Indirect Adaptive Control System.

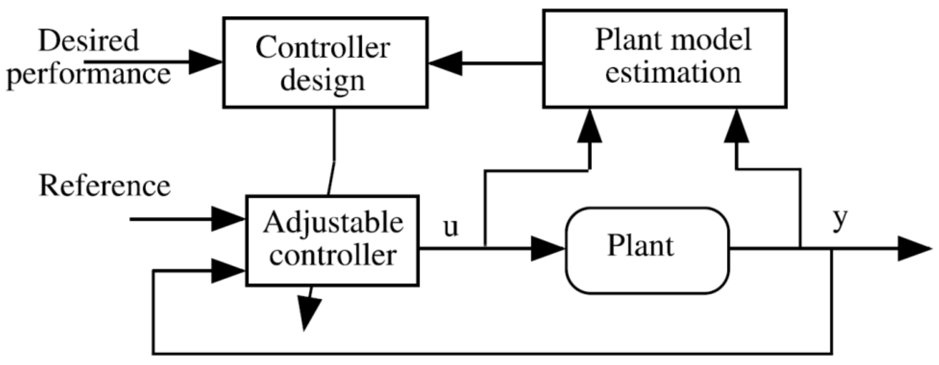


Fig. 2.8 Indirect adaptive Control [4]

As indicated in [4] the dynamic model of a plant can be identified from the input and output measurement of a plant obtained under an experimental protocol in open or in closed loop. Therefore, an adaptive control system can be seen as the identification or estimation of the dynamic plant model in real time in order to adjust the controller to achieve the desired performance.

The plant model in Fig 2.8 is estimated on- line from available inputs (u) and outputs(y) by the Plant Model Estimator and the controller's parameters are computed on-line (Controller design).

There are number of applications where indirect adaptive control techniques are used, like on Aircraft flight control systems as demonstrated on the tailless fighter aircraft: Lockheed Martin Tactical Aircraft Systems, where flight control system designers have successfully applied indirect adaptive control techniques to satisfy the requirement for cost-effective, robust, and reconfigurable control systems [15].

2.4 Adaptive Rollover Prevention

Any vehicle is prone to rollover therefore to ensure the UGVs reliability and survivability the enhancement of the vehicle's rollover resistance is crucial.

The first step in enhancing the rollover resistance is to determine the rollover index (RI), where Rollover index is defined as the variable that indicates the likelihood for vehicle to roll over [9], and it has to be determined or calculated in real time for appropriate action to be taken to prevent roll over.
When RI = 1 the vehicle is at a point of rollover the objective is to have system, which will prevent Rollover Index to get closer to this value (one) [9].
The proposed mechanisms which can be used to limit or prevent Rollover index from getting closer to the value of 1 is adaptive active suspension system and adaptive Speed control.

2.4.1 Adaptive Active Suspension

Active suspension automatically controls the vertical movement of the wheels relative to the vehicle body; this technology has been widely used by car manufactures to provide good car handling and ride quality, Fig 2.10 shows active suspension quarter model of a vehicle with Ms representing the vehicle body mass/sprung mass, Mu represent the wheel assembly/unsprung mass, K2 representing the spring, bs the shock absorber and Fa the actuator force.

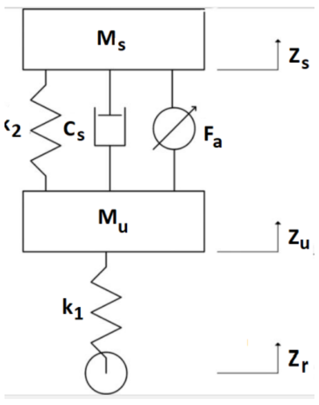


Fig. 2.9 Quarter model of vehicle suspension

Actuator force (Fa) in Fig. 2.9 which is the active component of the suspension system can be controlled in order to achieve a desired body roll dynamics.

This kind of similar techniques has already been used on passenger vehicles (e.g. Mercedes Benz Active body Control)

Another kind of active suspension is shown in Fig 2.10 which will be used in this dissertation to demonstrate the applicability of adaptive control system, but the adaptive control system technique can be used in any active suspension system design like the first one.

Equation (2.6) show that the lowering the roll angle lowers the rollover index |RI| and as on Fig 2.11 below, servo motors are used to adjust the vehicle suspension there varying the roll angle and which is used to maintain the rollover index within acceptable limits (-1<R<1)

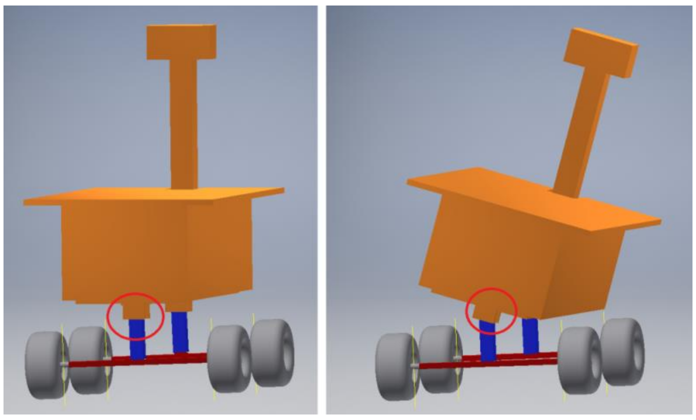


Fig. 2.10 Active suspension system

It is termed indirect adaptive active suspension system because the vehicle model is estimated online, which in turn is used to compute the rollover index and also used to adjust the suspension of the UGV in real time to counter act the increase in roll angle experienced beyond predetermined limits.

2.4.2 Adaptive Speed Control

The second mechanism to be used alongside indirect adaptive active suspension system is adaptive speed control.

It is termed adaptive speed control because the rollover Index is estimated online which in turn is used to adjust speed controller of the vehicle.

As it can be seen from equation (2.1) $a_y = \frac{{V_X}^2}{R}$ that the vehicle velocity is directly proportional to the vehicle lateral acceleration (a_y) which translates to lowering the speed reduces lateral acceleration which in turn as per equation (2.6) $RI = \frac{2h_{c.g}a_y cos\emptyset + 2gh_{c.g}sin\emptyset}{l_w g}$ reduces the prosperity for vehicle rollover.

2.5 Parameter Estimation

As stated in 2.4 that to accurately calculate the vehicle rollover index the roll angle has to be determined and but to measure roll angle is difficult and require expensive sensors, one has to estimate this parameter.

There are number of parameter estimation methods which can be used to estimate the roll angle, but two methods will be discussed in 2.5.1 and 2.5.2

2.5.1 Least Square Estimation

Since Karl Gauss proposed the technique of least squares around 1795 to predict the motion of planets and comets using telescopic measurements, least squares (LS) and its many variants have been extensively applied to solving estimation problems in many fields of application [16].

There are two basic forms of least squares methods which is the linear regression and the curve fitting method in which the curve fitting is defined as a process of constructing a curve or a mathematical function that has the best fit to a series of data points, for example if we have a linear equation (2.8).

$$y = mx + c \tag{2.8}$$

with two data points (x1, y1) and (x2,y2) available, then a straight line through the two data points with one point as reference and m being the slope from the difference between the two data points and C being the where the line cuts the y axis, therefore the equation is as illustrated in Fig 2.11

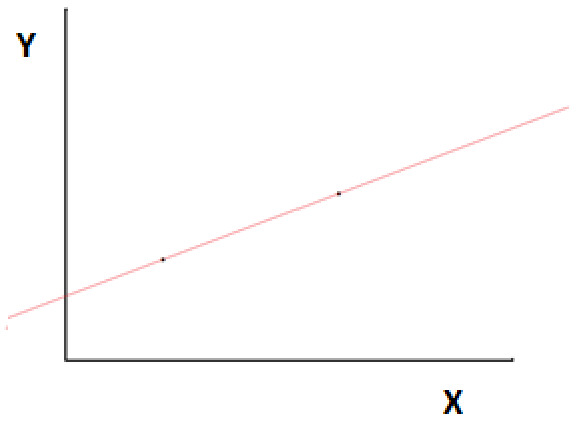


Fig. 2.11 Straight line graph

But if there are more than two data points which doesn't lie in a straight line as shown in Fig 2.8 it's not always possible to fit a straight line through all data points therefore, we need to find a straight line which fits the available data best, using Equation 2.9:

$$y = \widehat{m}x + \hat{c} \tag{2.9}$$

Where m and c are the unknowns and the deviations between known y and expected $(mx_n + c)$ at each data point is given as:

$$a_1 = (mx_1 + c) - y_1, \ \ a_2 = (mx_2 + c) - y_2, \ \ \ldots\ldots\ldots\ldots, a_n = (mx_n + c) - y_n$$

If m and c are selected in such a way that the deviations at each data point are zero will mean all data points are in a straight line as in Fig 2.11. but if the deviations at each data point are different as in Fig 2.12 therefore to obtain the best fit line we select m and c in such a way it minimize the square root of the sum of squares of the deviations (Least squares) meaning equation 2.10 must be as small as possible.

$$\sqrt{(a_1^2 + a_2^2 + \ldots + a_n^2)} \tag{2.10}$$

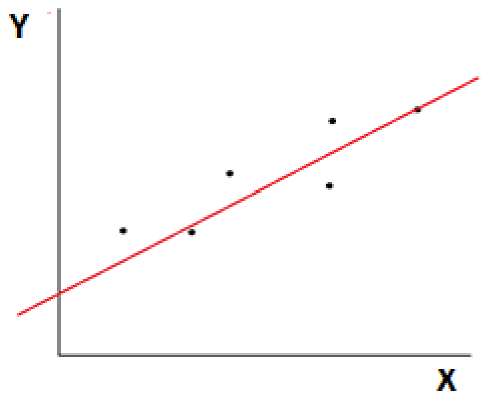


Fig. 2.12 best fit line

Assuming the following linear system:

$$y(t) = H(t)\theta(t) + e(t) \tag{2.11}$$

With

y = measurement output

H= known variables

θ= Unknown parameter

e =measurement error

Firstly equation (2.11) is written in regression form:

$$y(t) = H(t)^T\theta(t) + e(t) \tag{2.12}$$

After the N sampling periods equation 2.12 can be written in vector form:

$$\begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_N \end{bmatrix} = \begin{bmatrix} H_1^T \\ H_2^T \\ \vdots \\ . \\ H_N^T \end{bmatrix} \theta(t) + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ . \\ e_N \end{bmatrix} \tag{2.13}$$

Using Linear Least Squares algorithms, the unknown parameter θ is chosen in such a way to minimize the sum of squares errors [17].

$$J(\theta) = \sum_{t=1}^{N} e^2(t)$$

$$= e^T . e \tag{2.14}$$

Expanding 2.14

$$\begin{aligned} J(\theta) &= (y - H\theta)^T (y - H\theta) \\ &= y^T y - y^T H\theta - \theta^T H^T y + \theta^T H^T H\theta \\ &= y^T y - 2y^T H\theta + \theta^T H^T H\theta \end{aligned} \tag{2.15}$$

Taking derivative of 2.15:

$$\frac{\partial}{\partial\theta} J(\theta) = -2H^T y + 2H^T H\theta \tag{2.16}$$

Setting the derivative to zero:

$$\frac{\partial}{\partial\theta} J(\theta) = 0$$

$$0 = -2H^T y + 2H^T H\theta$$

$$H^T y = H^T H\theta \tag{2.17}$$

Provided that the inverse of $H^T H$ exist; therefore, the least square solution of 2.14 is gives as:

$$\theta(t) = [H(t)^T . H(t)]^{-1} . H(t)^T . y(t) \tag{2.18}$$

But this parameter estimation method uses batch processing where data is collected off-line prior to the estimation and it estimation has to be recomputed each time a new data point arrives which increases the computational requirement of the system therefore it not suitable for adaptive control where parameters have to be estimated in real-time/online and high computational requirement increases the cost factor of the system.

2.5.2 Recursive Least Square Estimation

On-line estimation of the parameters of a plant model or of a controller is one of the key steps in building an adaptive control system [4].

As indicated in 2.5.1 that the least square is not suitable for the UGV application, the recursive least squares which is derived from the Least Square method will be used due to its ability to recursively update the estimate in real time by updating the current estimate using the previous estimate without having to re-compute the entire data which reduces the computational requirement.

Derivation of the RLS is properly documented in many literatures [19] for explanatory purposes the RLS Algorithm is illustrated as follows:

First equation 2.11 is written in regression form:

$$y(t) = H^T(t)\theta(t) + e(t) \tag{2.19}$$

Where,

$$H^T(t) = [H_1(t), H_2(t) \dots H_n(t)];\ \theta = [\theta_1, \theta_2\ \dots \theta_n]^T \tag{2.20}$$

And the new update of the parameter estimate can be written as:

$$\theta(t) = \theta(t-1) + K(t).[y(t) - H^T(t)\theta(t-1)] \tag{2.21}$$

With the new update of the parameter estimate θ(t) at time (t) being determined by adding correction term to the previous parameter estimate θ(t-1) .

The correction term is formed by the update gain K(t), output measurement y(t), Regression vector $H^T(t)$ and the previous parameter estimate θ(t-1)

$$K(t).[y(t) - H^T(t)\theta(t-1)] \tag{2.22}$$

And update gain K (t) is calculated as:

$$K(t) = P(t).H(t) \tag{2.23}$$

Let; $P(t-1) = [H(t-1)^T.H(t-1)]^{-1}$ in equation 2.18

$$P(t-1)^{-1} = [H(t-1)^T.H(t-1)] \tag{2.24}$$

Therefore, at time (t)

$$P(t)^{-1} = [P(t-1)^{-1} + H(t)H(t)^T] \tag{2.25}$$

Using the matrix inversion lemma:

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(DA^{-1}B + C^{-1})^{-1}DA^{-1} \tag{2.26}$$

Equation (2.25) can be rewritten as:

$$P(t) = (P(t-1)^{-1})^{-1} - \frac{P(t-1)H(t)H(t)^T P(t-1)}{1 + H(t)^T P(t-1)H(t)} \tag{2.27}$$

Where P(t) is the covariance matrix calculated as:

$$P(t) = P(t-1) - \frac{P(t-1)H(t)H^T(t)P(t-1)}{\lambda + H^T(t)P(t-1)H(t)} \tag{2.28}$$

From equation 2.28
λ= represent the forgetting factor.

If one would like the recursive least squares algorithm older data to have less effect on the coefficient estimation, then λ is set to be less than 1 and if λ is set to 1 all-time steps data are of equal importance.

RLS algorithm involves two variables $P(t-1)$ and $\theta(t-1)$ therefore to initialize the recursions the initial values of these variables must be provided, typically if there is no previous information known about the parameter θ its common practice to use the following initial values: $\theta(0) = 0$ and $P(0) = \beta I$ where $\beta > 0$ "large number" and I denotes identity matrix.

Due to the recursive nature of this RLS algorithm it is the preferable estimation method to be used for online parameter estimation of the adaptive controller.

Chapter 3 Adaptive Control System Design

3.1 Introduction

This chapter provides a detailed design of the UGV's controller based on adaptive control algorithms, using recursive least squares estimation method; as it has been illustrated in chapter 2 that RLS is the most appropriate estimation method to be used for adaptive control system. The control algorithms are computed using Matlab Simulink® graphical programming environment for modelling, simulating and analysing multi-domain dynamical system.

3.2 Adaptive Controller Design Specifications

In order to satisfy the research objectives, the controller should be able to meet the following design specifications.

3.2.1 The adaptive control system must be stable

1. controller should be tuned to avoid system overshoot and oscillation.
2. The controlled output must be able to track the input signal as fast and as accurately as possible.

3.2.2 The adaptive controller must be Cost effective

1. Compact controller: Processor core, ROM, RAM, A/D converter and I/O channels on one package.
2. Controller should be programable using Matlab®: As the whole control system will be built, simulated and tuned in Matlab® Simulink.
3. Total controller hardware and programming software cost < R 2 500.00.

3.3 Vehicle Model

Before designing an adaptive controller, a vehicle Matlab® SimMechanics model was developed using UGV CAD modelled in Autodesk inventor® and exported to Matlab® SimMechanics using SimMechanics link 2nd Generation. Besides the vehicle model Importance in development of an adaptive controller it's also easier to understand, as the multibody vehicle parts are self-explanatory.

The model is developed in order to approximate the behaviour of the real UGV and help validated the applicability of the adaptive controller in real life situations.

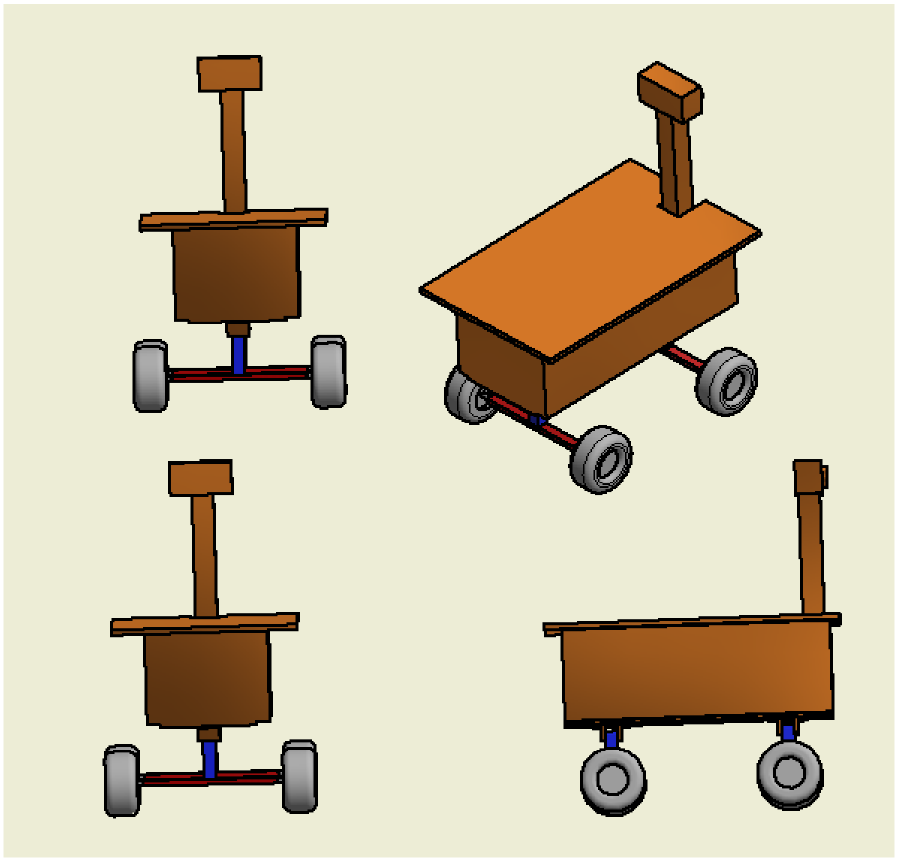


Fig. 3.1 UGV 3D Model

Fig 3.1 shows the 3D model of the 4 wheel UGV and Fig 3.2 is the UGV's Simulink model with basic inputs and output of the model listed in table 3.1.

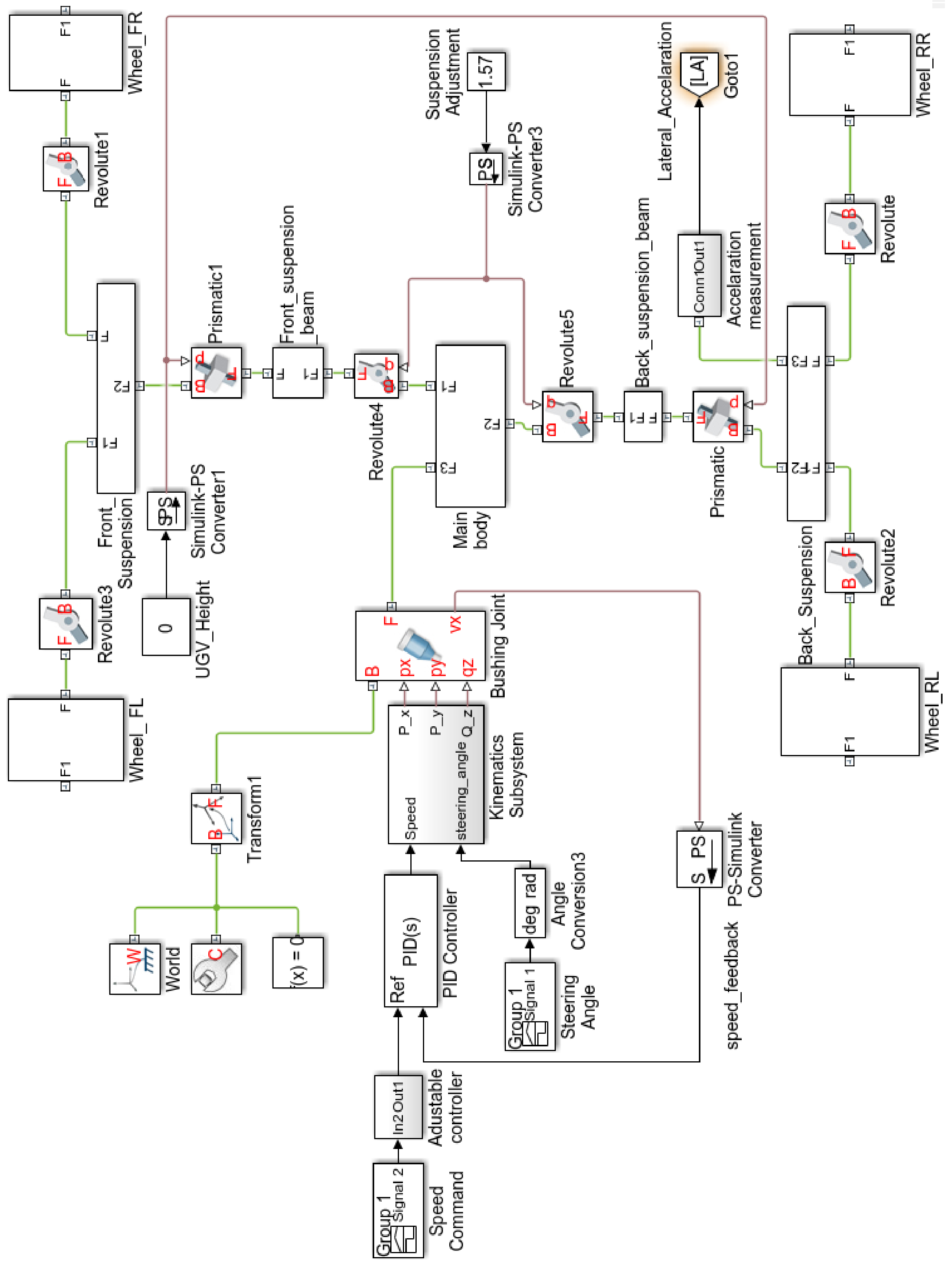


Fig. 3.2 Simulink UGV Model

Table 3.1 Simulink UGV input/output variables

INPUT		DISCRIPTION
1	Steering Angle	Gives the UGV heading and trajectory
2	Vehicle Speed Command	Desired vehicle speed (speed set-point to PID controller)
3	Suspension Angle	used for suspension adjustment (Vehicle body rotation)
OUTPUT		**DISCRIPTION**
1	Lateral acceleration	Lateral acceleration measured at unsprug mass of the back suspension
2	Speed	Measured vehicle speed

The steering angle and the vehicle speed inputs are controlled remotely by the UGV driver, but the speed is operated within a specific range as determined automatically by the adaptive controller depending on the predetermined desired rollover index performance parameter limits, while the suspension angle will be automatically adjusted by the adaptive controller depending on the predetermined desired roll angle limits.

In order to run the model in Matlab Simulink simulation environment a dynamic kinematic model of the vehicle has to be developed as per the following mathematical equations:

The first one is the heading angle in which for the purpose of simulations in chapter 4, is derived in way that the vehicle will follow the same path trajectory regardless of the speed and time it takes to negotiate that specific path but under normal operation the trajectory will be provided by the UGV driver remotely.

The input speed is used to obtain the distance the UGV travelled as per equation 3.1

$$d(t) = \int_{t0}^{t} \omega_{in}\,(t)dt \qquad (3.1)$$

The distance is then fed in to the input of a lookup table block which gives out an output heading angle based on the predefined table (e.g. when distance reach 10m the vehicle must turn -5 degrees)

Then the output of the lookup table is fed to a low pass Bessel filter which smooths out the heading angle waveform.

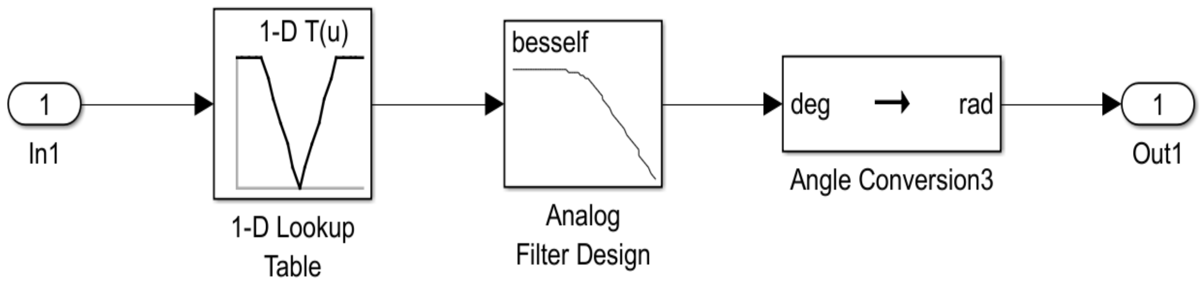
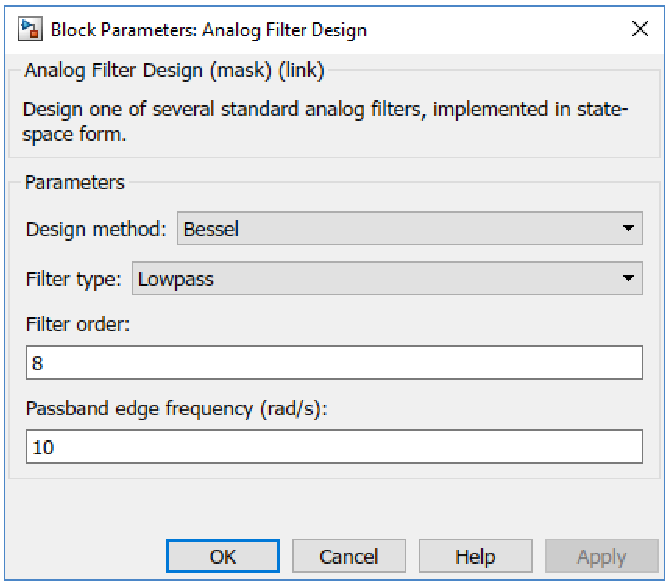


Fig. 3.3 Vehicle heading angle


Block Parameters: Analog Filter Design

Analog Filter Design (mask) (link)

Design one of several standard analog filters, implemented in state-space form.

Parameters

Design method: Bessel

Filter type: Lowpass

Filter order:

8

Passband edge frequency (rad/s):

10

Fig. 3.4 Bessel filter parameters

Then the output heading angle along with the vehicle speed is fed to the Kinematics subsystem in Fig 3.5 which is translated into vehicle motion as per the following mathematical equations.

$$\omega_x = \omega_{in} \sin \delta \tag{3.2}$$

$$\omega_y = \omega_{in} \cos \delta \tag{3.3}$$

The vehicle x and y position are as per equation 3.4 and 3.5 respectively:

$$P_x(t) = \int \omega_x\,(t)dt \tag{3.4}$$

$$P_y(t) = \int \omega_y\,(t)dt \tag{3.5}$$

And the vehicle Z rotation:

$$Q_z = \delta \tag{3.6}$$

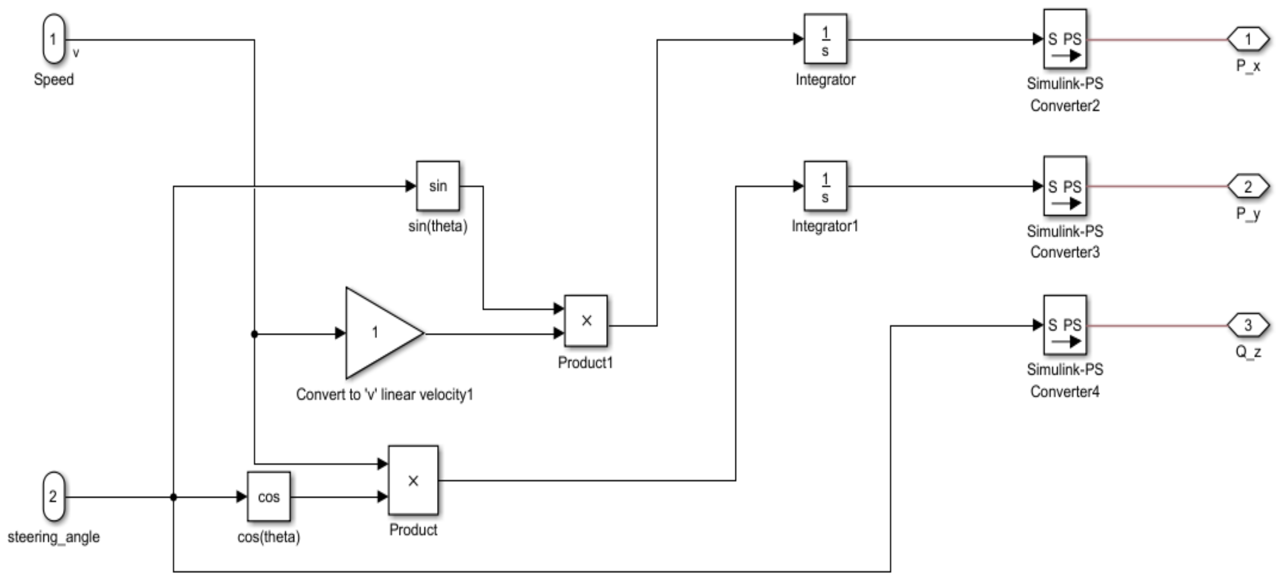


Fig. 3.5 kinematics subsystem

3.4 Adaptive Control

The main objective of the controller is to autonomously maintain the rollover index of the UGV within acceptable limits: $(-1 < RI_d < 1)$ and the roll angles $(-4^0 < \o_d < 4^0)$ which will translate into rollover prevention but these limits are user defined as the above are for explanatory purposes.

In order to achieve the above desired parameters, adaptive control algorithms have to be developed.

3.4.1 Overview

Fig 3.6 presents the UGV adaptive control system architecture based on recursive least squares estimation of unknown parameters, the estimation is done in order to compute the required commands in real time to adjust suspension actuators and to adjust the PID controller's input speed (set-point).

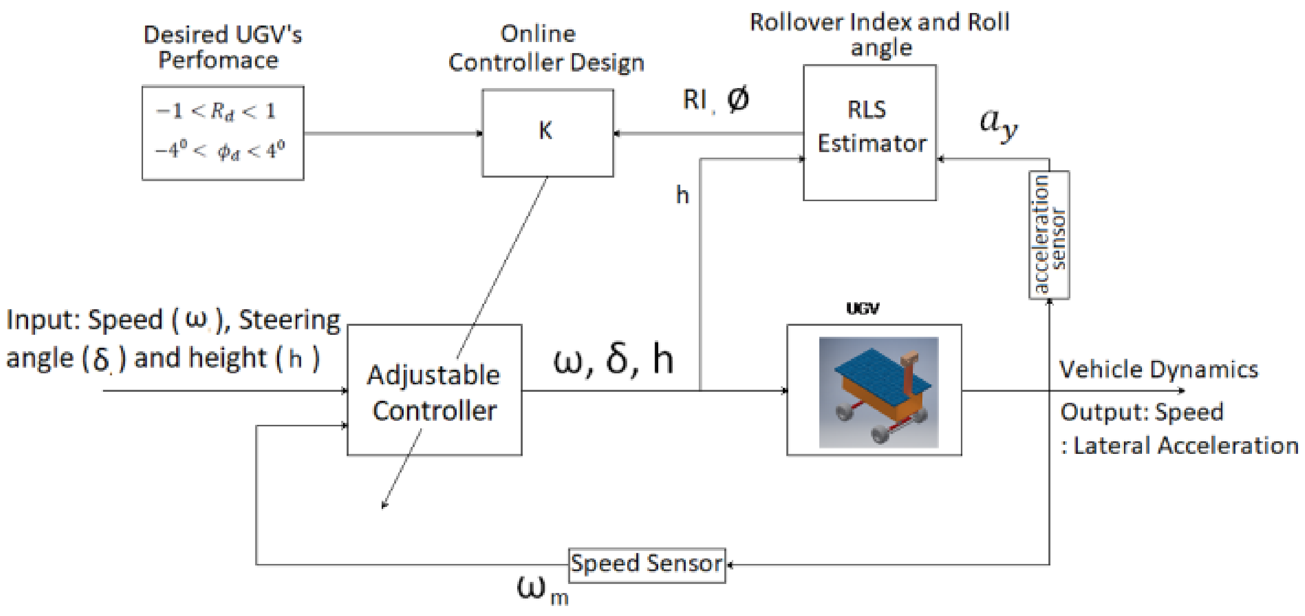


Fig. 3.6 Indirect Adaptive Control System architecture

The adaptive control system in Fig 3.6 operates as follows:

Step 1: The operator will give the vehicle speed, the steering angle (vehicle heading) and vehicle height (which determines the vehicle height of centre of gravity) commands to the Adjustable Controller

Step 2: The adjustable controller will provide the UGV heading, vehicle height and speed using a set electric motors and servo motors

Step 3: Output of the UGV is the vehicle dynamics which includes the speed and lateral acceleration.

Step 4: Then the RLS Estimator estimates the rollover index and Roll angle using measured lateral acceleration and the UGV's height of centre of gravity (c.g.)

Step 5: The Estimated (RI, Ø) together with the predetermined desired UGV performance builds an online controller.

Step 6: The output/gain of the online controller is fed to the adjustable controller to limit the UGV speed within safe predefined operational values.

Fig 3.6 can be better explained by a flow diagram in Fig 3.7.

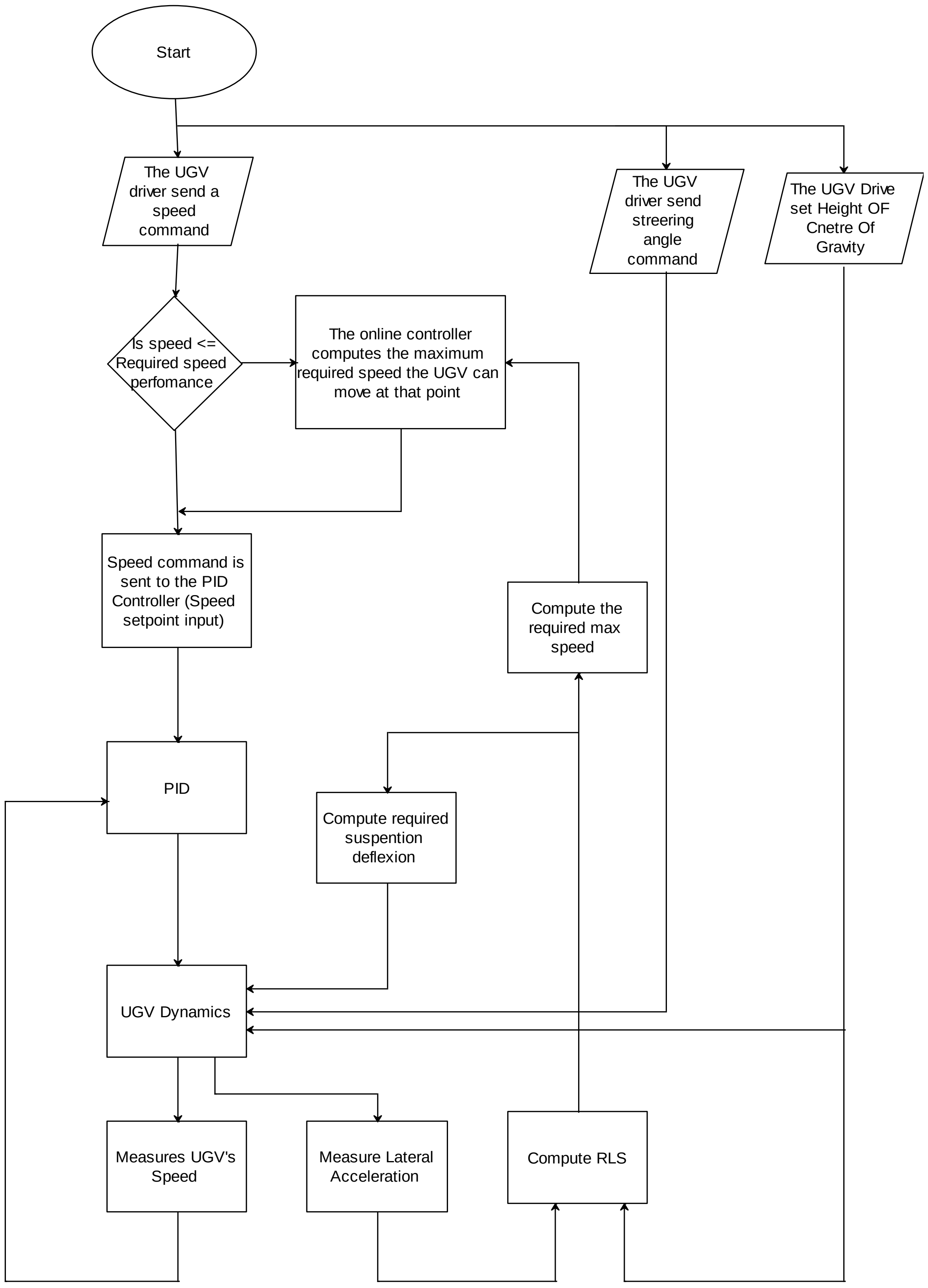

Start
The UGV driver send a speed command
The UGV driver send streering angle command
The UGV Drive set Height OF Cnetre Of Gravity
Is speed <= Required speed perfomance
The online controller computes the maximum required speed the UGV can move at that point
Speed command is sent to the PID Controller (Speed setpoint input)
Compute the required max speed
PID
Compute required suspention deflexion
UGV Dynamics
Measures UGV's Speed
Measure Lateral Acceleration
Compute RLS

Fig. 3.7 Indirect Adaptive Control System flow diagram

Fig 3.8 shows the Matlab Simulink® model of the system based on the flow diagram in Fig 3.7

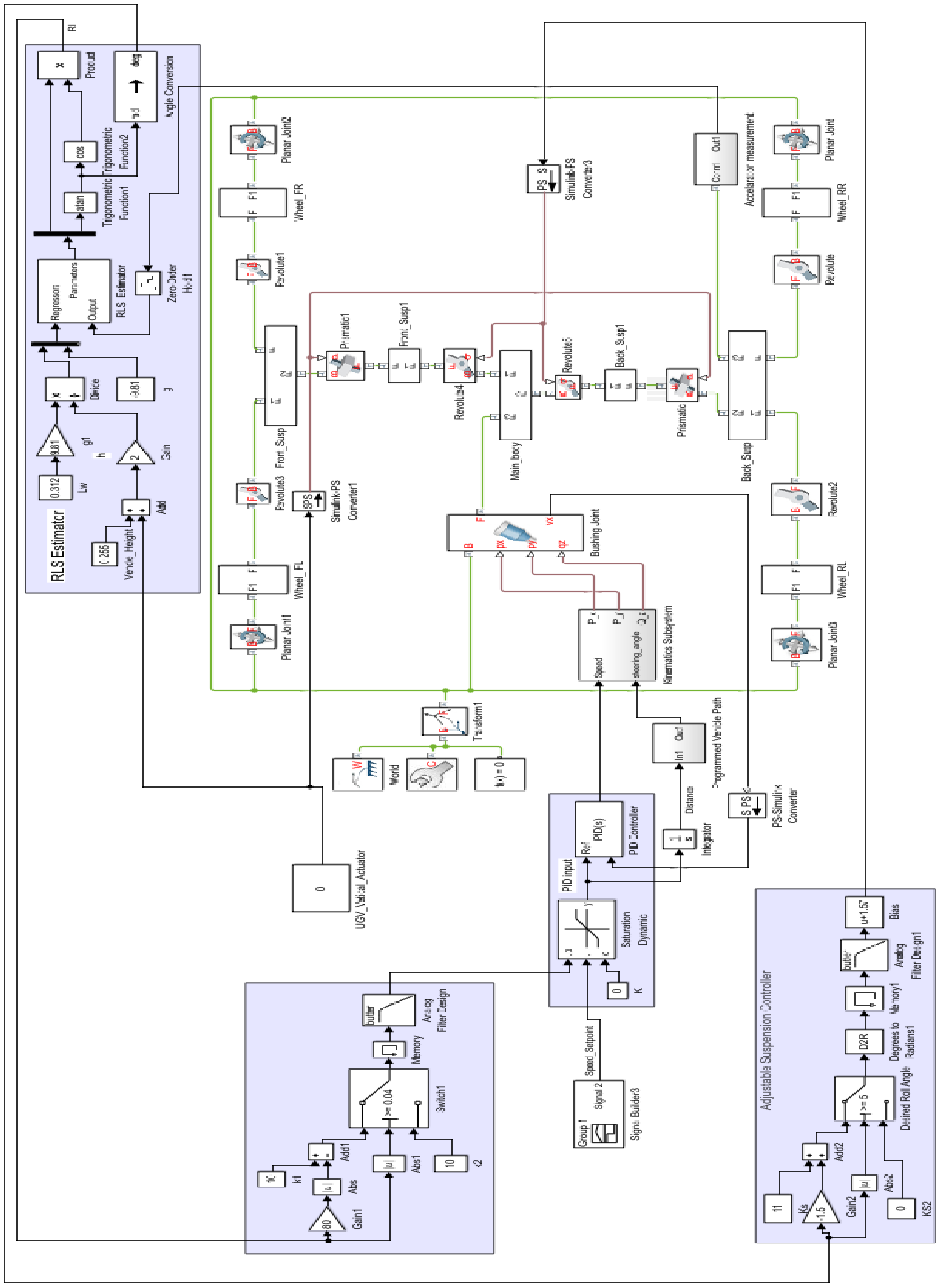


Fig. 3.8 Indirect Adaptive Control System Matlab Simulink® model

The whole adaptive operation can be summarized into two main designs: RLS Estimator Design (online estimation of the rollover index and Roll-Angle) and online control design (on-line computation of the controller parameters based on the current estimated rollover index and Roll-Angle).

3.4.2 RLS Estimator Design

As discussed in chapter 2 the unknown parameters to be estimated is the roll angle and rollover Index, using the known input parameter $h_{c.g}$ (height of centre of gravity of the vehicle) and the output a_y (lateral acceleration) parameter; in which the RLS algorithm is developed by looking at the vehicle lateral dynamics, using equation (2.6):

Therefore, to estimate the roll angle we need lateral acceleration measured at un-sprung mass therefore equation (2.6) can be written as:

$$a_y = \frac{l_w g RI}{2h_{c.g}\cos\emptyset} - g\tan\emptyset \tag{3.7}$$

Then equation (3.1) is rearranged in regression form:

$$\mathrm{y(t)} = H^T(t)\theta(t) \tag{3.8}$$

With the measured acceleration

$$\mathrm{y(t)} = a_y(t) \tag{3.9}$$

Known variables

$$H^T(t) = \begin{bmatrix} \frac{l_w g}{2h_{c.g}} & - & g \end{bmatrix} \tag{3.10}$$

Unknown parameter (Rollover index and roll angle)

$$\theta(t) = \begin{bmatrix} \frac{RI}{\cos\emptyset} \\ \tan\emptyset \end{bmatrix} \tag{3.11}$$

Using RLS algorithm the unknown parameter at time (t) is given as equation (2.21):

$$\theta(t) = \theta(t-1) + K(t).[y(t) - H^T(t)\theta(t-1)]$$

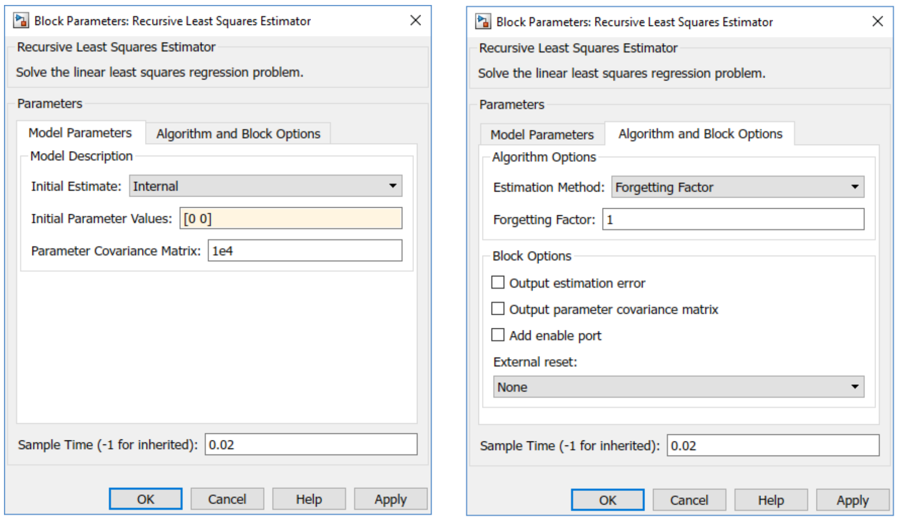
Substituting equation 2.21 with equation 3.9, 3.10 and 3.11:

$$\begin{bmatrix} \frac{RI}{\cos \emptyset} \\ \tan \emptyset \end{bmatrix} = \begin{bmatrix} \frac{RI}{\cos \emptyset} \\ \tan \emptyset \end{bmatrix}(t-1) + K(t).[a_y - \left[\frac{l_w g}{2h_{c.g}} \;-\; g\right]. \begin{bmatrix} \frac{RI}{\cos \emptyset} \\ \tan \emptyset \end{bmatrix}(t-1)] \tag{3.12}$$

$$K(t) = P(t).H(t)$$

$$P(t) = P(t-1) - \frac{P(t-1)H(t)H^T(t)P(t-1)}{1+H^T(t)P(t-1)H(t)} \tag{3.13}$$

All the UGV recursive least squared algorithms defined above are executed using Simulink RLS estimator block with its parameters set as in Fig 3.9


Block Parameters: Recursive Least Squares Estimator

Recursive Least Squares Estimator

Solve the linear least squares regression problem.

Parameters

Model Parameters | Algorithm and Block Options

Model Description

Initial Estimate: Internal

Initial Parameter Values: [0 0]

Parameter Covariance Matrix: 1e4

Sample Time (-1 for inherited): 0.02

Block Parameters: Recursive Least Squares Estimator

Recursive Least Squares Estimator

Solve the linear least squares regression problem.

Parameters

Model Parameters | Algorithm and Block Options

Algorithm Options

Estimation Method: Forgetting Factor

Forgetting Factor: 1

Block Options

☐ Output estimation error

☐ Output parameter covariance matrix

☐ Add enable port

External reset: None

Sample Time (-1 for inherited): 0.02

Fig. 3.9 RLS Estimator block

On the model parameters the initial estimate which is the initial guess of the values of the parameters to be estimated is selected to internal estimate with parameter values of [0,0] meaning $\frac{RI}{\cos \emptyset} = 0$ and $\tan \emptyset = 0$ when the UGV is powered on or the control system is initialized.

The parameter covariance matrix which defines the amount of uncertainty in the initial guess is set to 1e4 (Very high value) since there is no previous information known about the parameter θ.

On the algorithm and block options the estimation method is selected to be a forgetting factor which specifies the measurement window relevant for parameter estimation in which in this case a forgetting factor of 1 is selected meaning there is no forgetting.

Fig 3.10 shows the Matlab Simulink® model representation of the recursive least estimation of the UGV adaptive controller, the simulation to prove its applicability will be dealt with in chapter 4

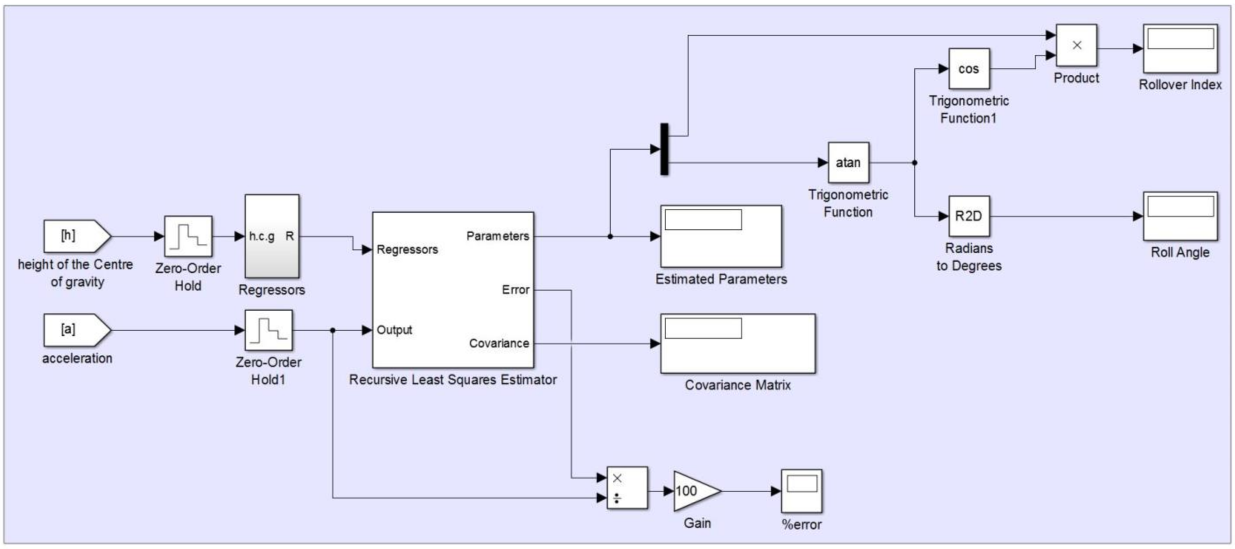


Fig. 3.10 RLS Estimator Matlab Simulink® model

The terms ($\frac{l_w g}{2h_{c.g}} - g$) in equation 3.10 are the model regressors which are the input to the recursive least squares block in which the estimated parameters ($\frac{RI}{\cos\emptyset}, \tan\emptyset$) are given out,

With:

$$\theta_1(t) = \frac{RI}{\cos\emptyset} \tag{3.14}$$

$$RI(t) = \cos\emptyset(t)\,\theta_1(t) \tag{3.15}$$

And
$$\theta_2(t) = \tan\emptyset \tag{3.16}$$

$$\emptyset(t) = \tan^{-1}\theta_2(t) \tag{3.17}$$

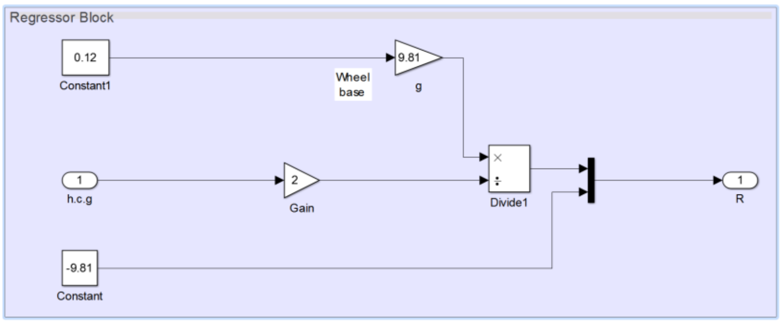


Fig. 3.11 Regressor Block

3.4.3 Online controller design: Speed Control

Online controller is an integral part of the adaptive controller when the controller is designed on line from the estimated variables (RI and Roll angle).

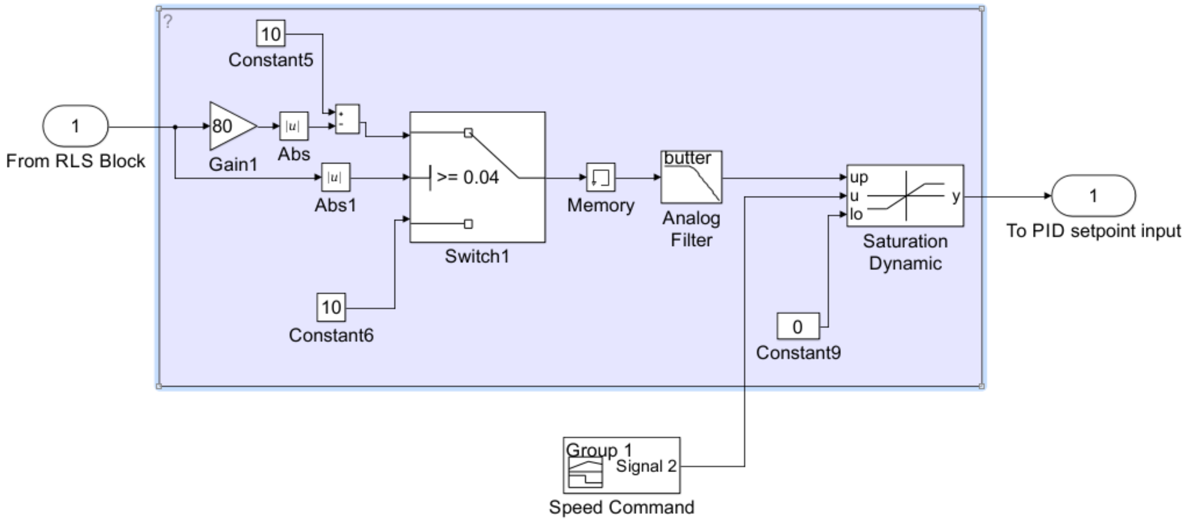


Fig. 3.12 Simulink online speed controller Block

Fig 3.12 shows the model representation of the online speed controller block built in Simulink with:

1. The roll angle (RI) estimation from RLS estimator block being the input to online controller.
2. Then a switch block is used to check if RI > = 0.04 then the output of the switch will be = (90- (80XRI) and if RI < 0.04 the output will be = 10
3. Then the output from the switch is fed into memory block (which is used to provide initial parameter when the system initialises which is set to 2 in this case)
4. The output of the memory block is fed to low pass Butterworth filter which cut off high frequency responses beyond 1 rads/s to prevent un-warranted system response

5. The output of the Butterworth filter is fed into the upper limit of the saturation dynamic block there by setting the upper limit (maximum speed the UGV can travel at that instant) and if the command speed is higher that the upper limit therefore the output of the saturation dynamic block is set to upper limit
6. Then the output of the saturation dynamic block is fed to the PID controller setpoint input

3.4.4 Online controller design: Suspension control

As stated in 3.4.3 that an online controller integral part of the adaptive controller when the controller is designed on line from the estimated variables (RI and Roll angle).

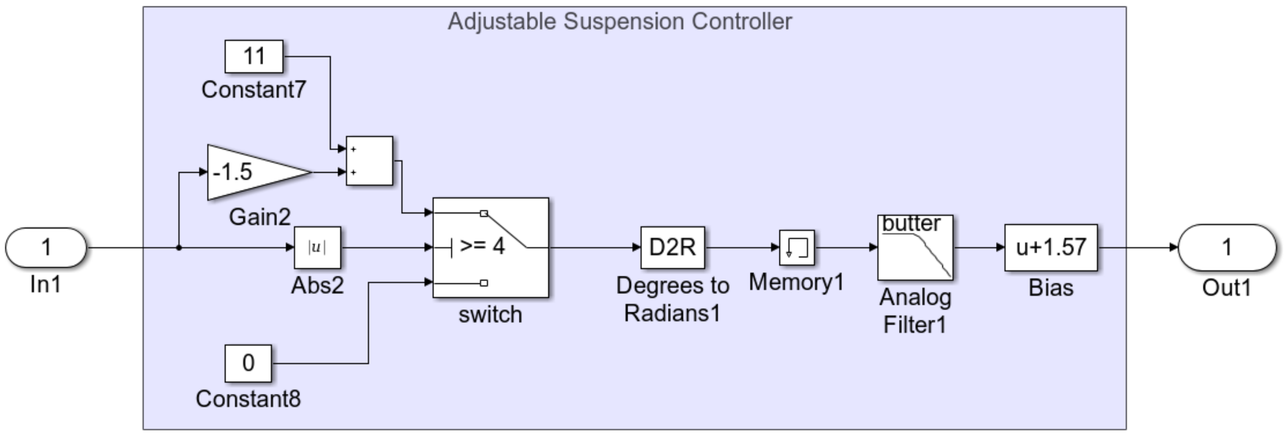


Fig. 3.13 Simulink online suspension controller Block

Fig shows the model representation of the online speed controller block built in Simulink with:

1. The roll angle (Ø) estimation from RLS estimator block being the input to online controller.
2. Then a switch is used to check if Ø > = 4 degrees then the output of the switch will be = (11+ (-1.5XØ) and if Ø < 5 the output will be = 0
3. Then the output from the switch is fed into memory block (which is used to provide initial parameter when the system initialises which is set to 1.57 in this case) then the output of the memory block is fed to low pass Butterworth filter which cut off high frequency responses beyond 1 rads/s to prevent un-warranted response
4. The output of the Butterworth filter is fed into bias block (which is used to set the angle of the actuators)

3.4.5 Indirect Adaptive Active Suspension System design

An indirect adaptive control system will be employed into the vehicle to automatically adjust the suspension to prevent the vehicle from rolling over

It is termed indirect adaptive control as the vehicles estimated Rollover index and the roll angle are computed online/real time from acceleration measurements and the vehicle height of centre of gravity; using RLS techniques the control parameters (K) are computed online from the Estimated Roll Angle and desired UGV's performance $(-4^0 < \phi_d < 4^0)$

The indirect adaptive control system is as per Fig 3.14 below:

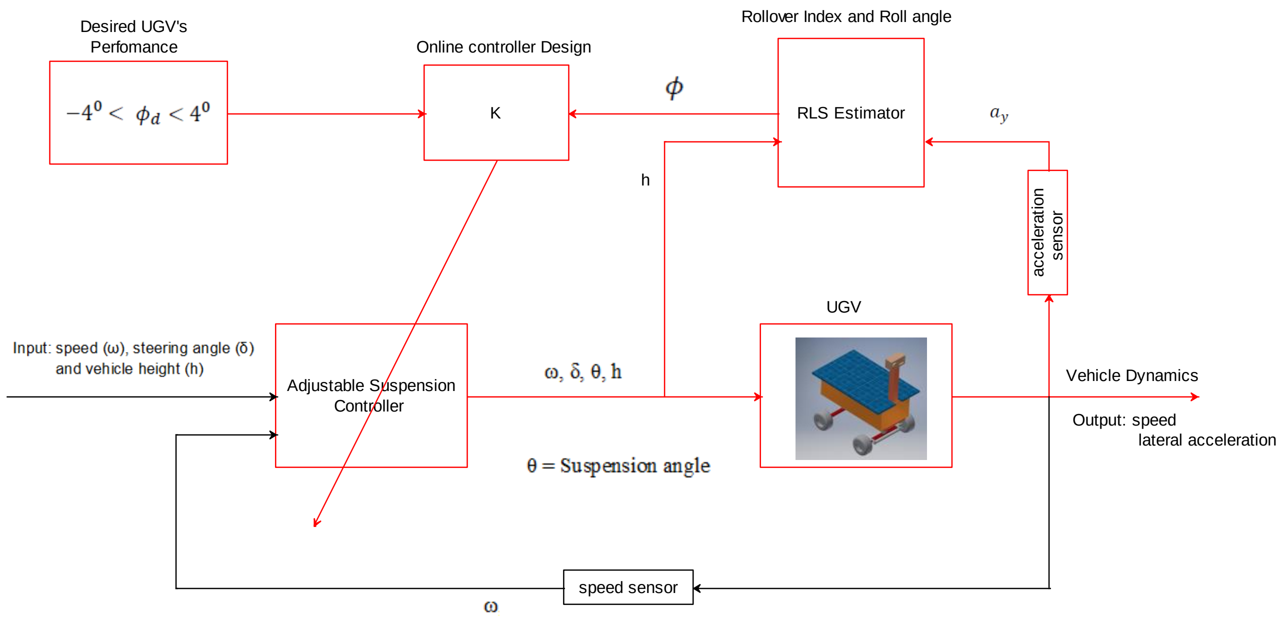


Fig. 3.14 Indirect adaptive active suspension control

System Description

1. At start up the on-board vehicle controller receive a command to reset or set the vertical actuators position in-turn it set the UGV height of the centre of gravity $(h_{c.g})$ to a specific position.

2. The accelerometer mounted at the vehicle unsprung mass (Back suspension) measures lateral acceleration of the vehicle.
3. Then the acceleration measurement signal is fed to the RLS estimator which computes/updates the Estimated rollover index (RI(t)) and roll angle ($\emptyset(t)$) at each sampling time from the lateral acceleration measurement and height of the centre of gravity command ($h_{c.g}$).
4. Then a controller is designed online using the UGV's desired performance ($-4^0 < \phi_d < 4^0$) and the estimated roll angle to give a control signal k.
5. Then the gain k is fed to the adjustable controller which in turn adjusts the suspension represented by a revolute joint on the Simulink vehicle model in order to counter act the roll motion.
6. The controller adjusts the correct side depending on the sign of the roll angle (+/-)

Fig 3.14 shows the created Matlab Simulink® model representation Indirect adaptive active suspension control of the UGV adaptive controller, the simulation of the system is dealt with in chapter 4.

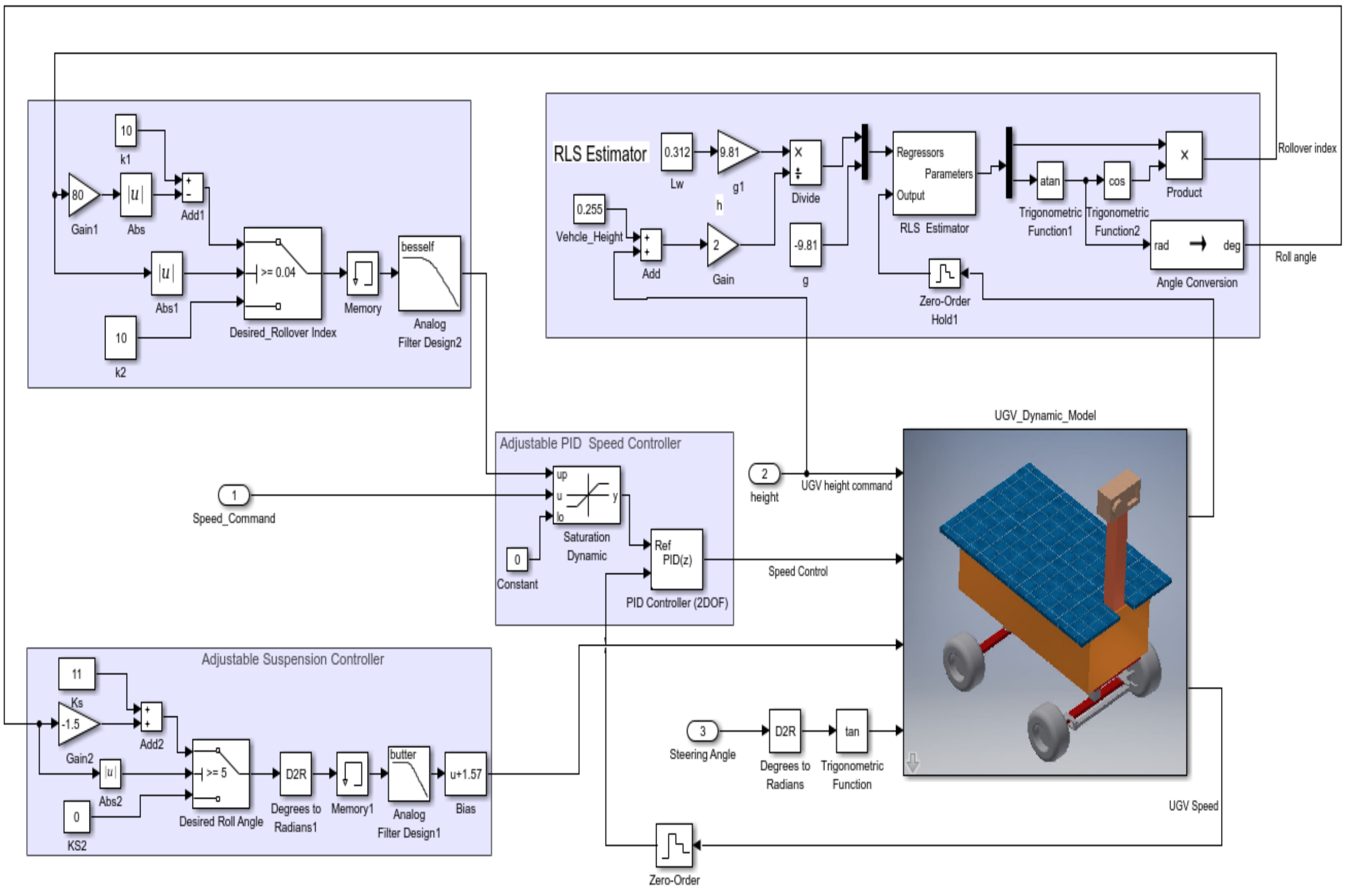
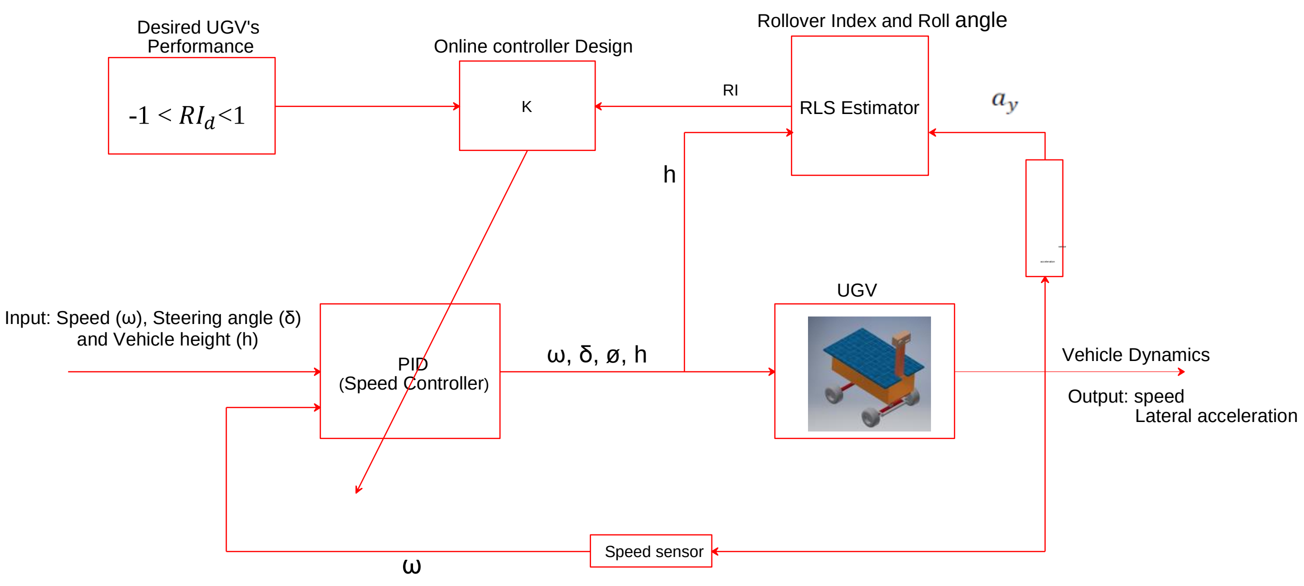


Fig. 3.15 Indirect adaptive active suspension control Matlab Simulink® mode

3.4.6 Indirect Adaptive speed control

An indirect adaptive speed control system will be used to limit the speed in order to reduce the prosperity for rollover to occur.

It is termed indirect adaptive control as the vehicles Roll over index (Estimated) is computed online/real time from acceleration measurements and the controller (K) is computed online from the Estimated Roll over index and desired UGV's performance in order to appropriately adjust the adjustable speed controller.
Indirect adaptive speed control system is as per diagram in Fig 3.15.



Desired UGV's Performance
$-1 < RI_d < 1$
Online controller Design
K
RI
Rollover Index and Roll angle
RLS Estimator
a_y
h
UGV
Input: Speed (ω), Steering angle (δ) and Vehicle height (h)
PID (Speed Controller)
ω, δ, ø, h
Vehicle Dynamics
Output: speed Lateral acceleration
Speed sensor
ω

Fig. 3.16 Indirect Adaptive Speed Control Diagram

1. At start a speed command is fed to Speed PID controller (set-point)
2. The PID controller computes a speed control signal which controls the speed of the vehicle from the set-point and feedback signal (from speed sensor measurements)
3. As the vehicle moves an accelerometer mounted at the vehicle back suspension (unsprung mass) measures the vehicle lateral acceleration.
4. Then the acceleration measurement signal (a_y) and the vehicle height of centre of gravity $(h_{c.g})$ are fed to the RLS estimator which computes/updates the Estimated rollover index (RI(t))
5. Then the online controller is designed by computing the desired performance $(-1 < RI_d < 1)$ and comparing it to the estimated rollover index.
6. Then the gain K is computed and used to set max limit to the PID speed controller set-point

Fig 3.16 shows the created Matlab Simulink® model representation Indirect adaptive active speed control of the UGV adaptive controller, the simulation of the system is dealt with in chapter 4

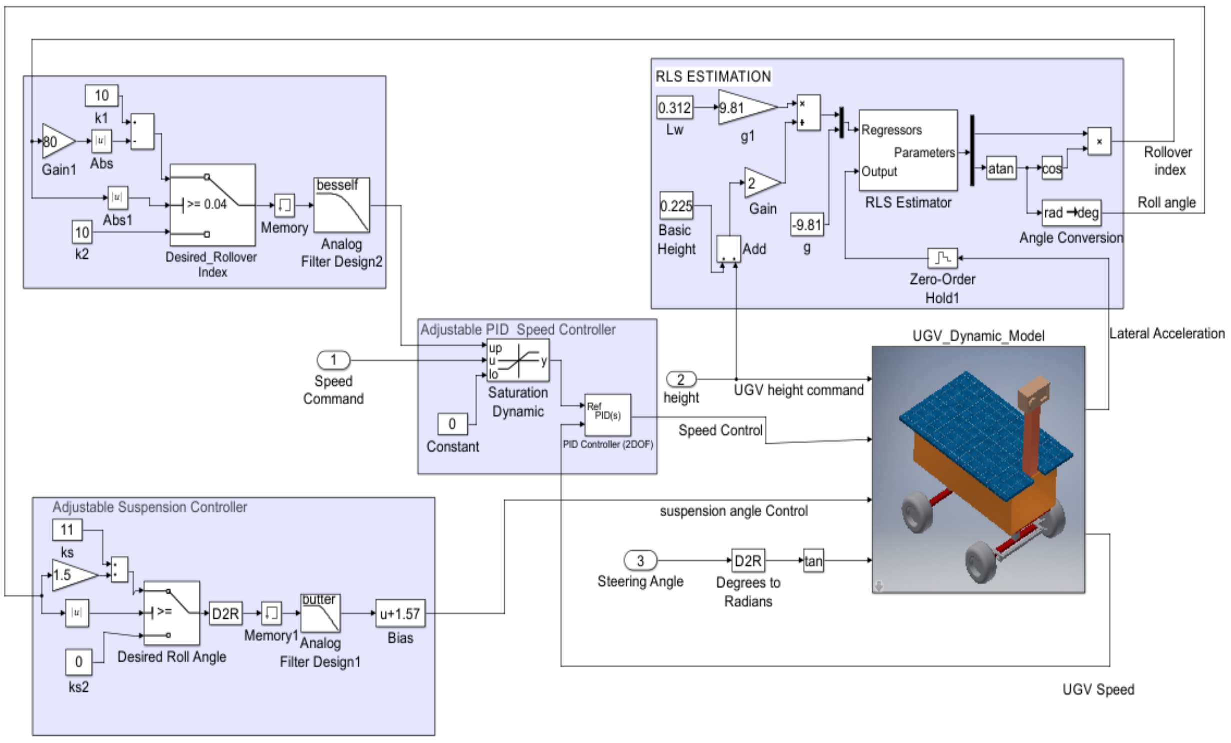


Fig. 3.17 Indirect Adaptive Speed Control Matlab Simulink® model

3.5 Controller Tuning

In order to meet the design specifications PID controller tuning is done using Matlab Simulink tuning method to automatically tune the controller gains based on the vehicle model in Simulink

Firstly, the control system is built in Simulink as per Simulink model in Fig 3.17 with a step input

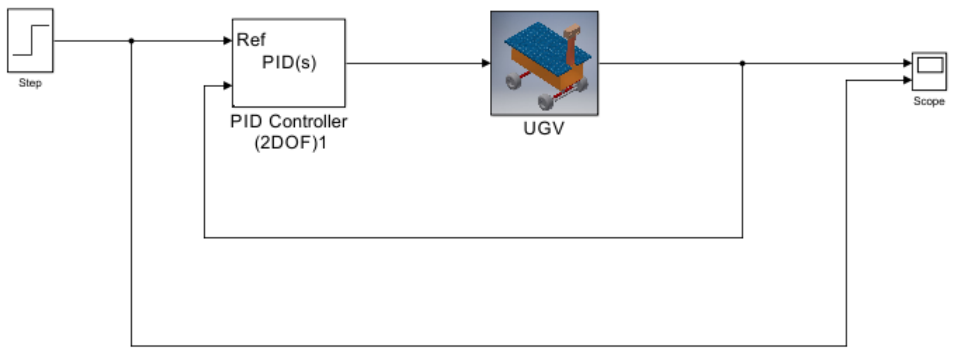
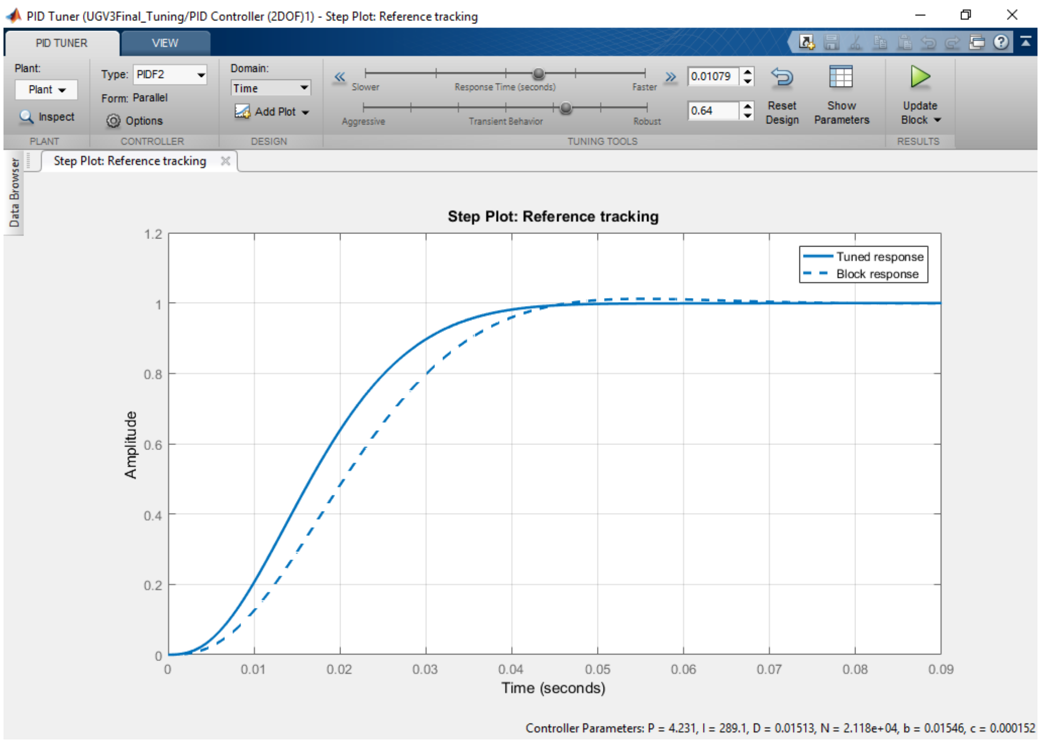


Fig. 3.18 Simulink Model

Then tune the controller using Simulink PID tuner as shown in Fig 3.18 by using the response time and transient behaviour in order to achieve the desired output (0 overshoot, no oscillation)

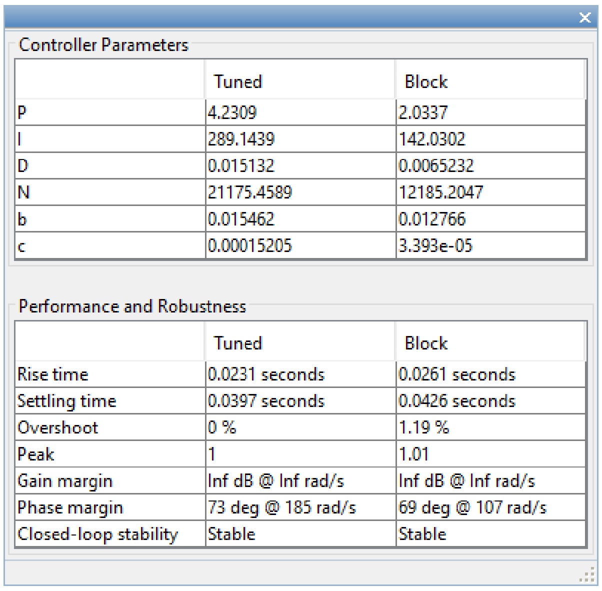
Fig 3.18 show a step plot with the desired tune response shown by solid blue line and the dotted blue line indicating the original response (untuned response) and Fig 3.19 show the controllers parameters and performance parameters (stability margins)


PID Tuner (UGV3Final_Tuning/PID Controller (2DOF)1) - Step Plot: Reference tracking

Type: PIDF2
Form: Parallel
Domain: Time
Response Time (seconds): 0.01079
Transient Behavior: 0.64

Controller Parameters: P = 4.231, I = 289.1, D = 0.01513, N = 2.118e+04, b = 0.01546, c = 0.000152

Fig. 3.19 PID Tuning step reference tracking


Controller Parameters

	Tuned	Block
P	4.2309	2.0337
I	289.1439	142.0302
D	0.015132	0.0065232
N	21175.4589	12185.2047
b	0.015462	0.012766
c	0.00015205	3.393e-05

Performance and Robustness

	Tuned	Block
Rise time	0.0231 seconds	0.0261 seconds
Settling time	0.0397 seconds	0.0426 seconds
Overshoot	0 %	1.19 %
Peak	1	1.01
Gain margin	Inf dB @ Inf rad/s	Inf dB @ Inf rad/s
Phase margin	73 deg @ 185 rad/s	69 deg @ 107 rad/s
Closed-loop stability	Stable	Stable

Fig. 3.20 Tuned controllers' parameters

Fig 3.20 shows the comparison between the step input and the output of the closed loop control system in Fig 3.17

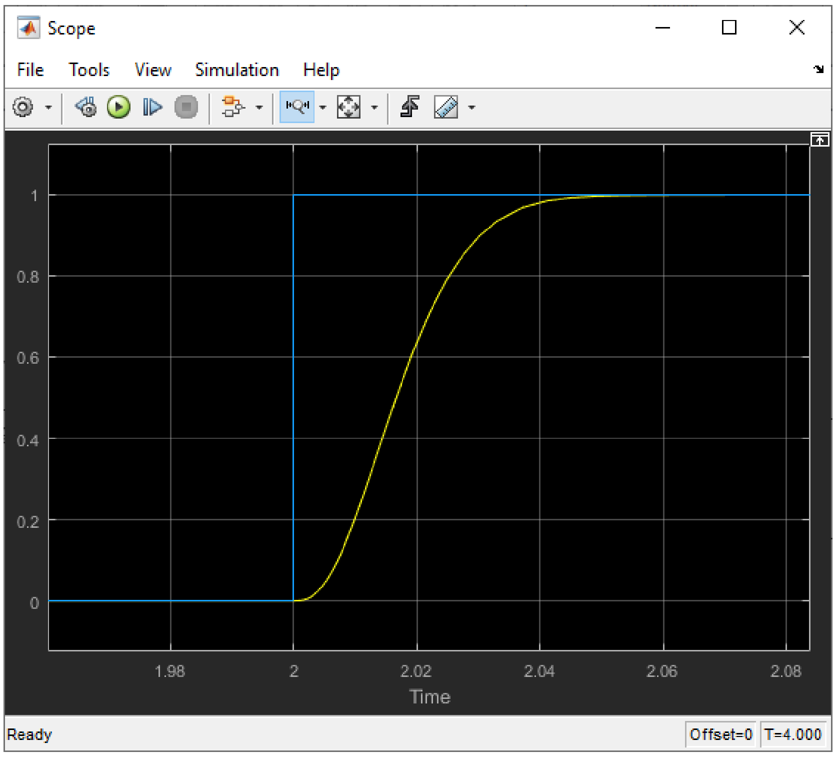


Fig. 3.21 Input step reference VS output step response

3.6 Controllers Cost

Below is the controller's hardware suitable for the required application

Item	Cost
1. Arduino Mega Microcontroller: ATmega2560-16AU Operating Voltage: 5V Input Voltage (recommended): 7-12V Input Voltage (limits): 6-20V Digital I/O Pins: 54 (of which 15 provide PWM output) Analog Input Pins: 16 DC Current per I/O Pin: 40 mA DC Current for 3.3V Pin: 50 mA Flash Memory: 256 KB of which 8 KB used by bootloader SRAM: 8 KB EEPROM: 4 KB Clock Speed: 16 MH	**R 500.00**
2. MATLAB and Simulink Student Suite With: MATLAB Coder, Simscape. Simscape Multibody, Simulink 3D Animation and Simulink Coder Add ons	**R 1 500.00**
Total Cost	**R 2 000.00**

3.7 Conclusion

In this chapter an adaptive controller algorithm and techniques is developed using Matlab Simulink and can be uploaded to an Arduino board which is capable of processing the control algorithms and the closed loop PID control is tuned using Simulink PID tuner to meet the desired design specification.

The cost of the selected controller is also below the budgeted amount of R 2500.00

Chapter 4 Adaptive control Simulations

In this chapter the designed adaptive control algorithms in chapter 3 will be evaluated using Matlab Simulink® simulations which will be sub-divided into two sections: Test-Rig Tilt table UGV adaptive control performance evaluation and UGV adaptive control performance evaluation using Simulink to simulate real world applications.

Both adaptive speed control and adaptive suspension control will be evaluated separately in order to determine each one's applicability.

4.1 Test-Rig Tilt table performance evaluation

A Test-Rig Tilt table was modelled in Inventor and imported along with the UGV model into Simulink environment using Simscape Multibody Link, with Fig 4.1 showing visualization of the imported UGV on test rig.

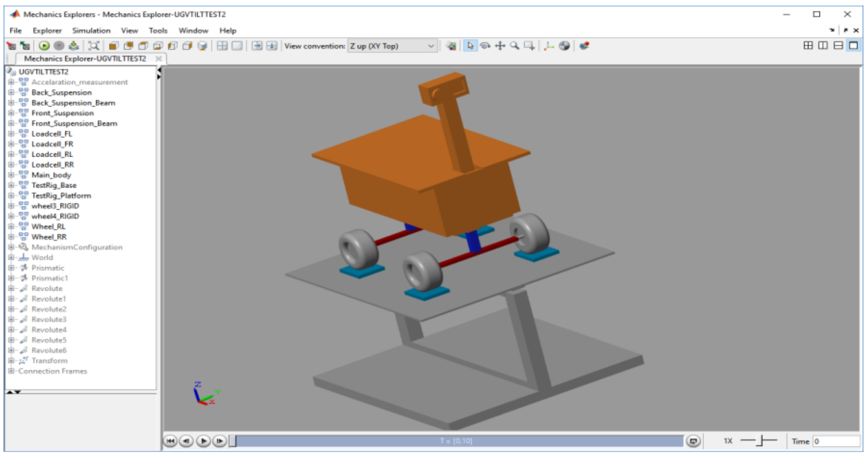


Fig. 4.1 UGV on the test Rig (SimMechanics: mechanics explorer)

4.1.1 Test Rig simulation without adaptive controller.

Initially the test-rig simulation was conducted without adaptive controller by gradually tilting the table from 0 to 85°

With Table 4.1 showing inputs to the model:

Table 4.1: UGV Test Rig Model input variables

INPUT		Value	Description
1	Steering Angle	0	The vehicle is straight
2	Vehicle Speed	0	The vehicle is not moving
3	Suspension Angle	0	The is no deflection on the suspension
4	Test-Rig Tilt Angle	0- 85°	The Test-Rig tilt table moves from 0° to 85°

Fig 4.2 provides the UGV simulink model on the test-rig with four loadcells (Vertical force (Fz) measurement sensors).

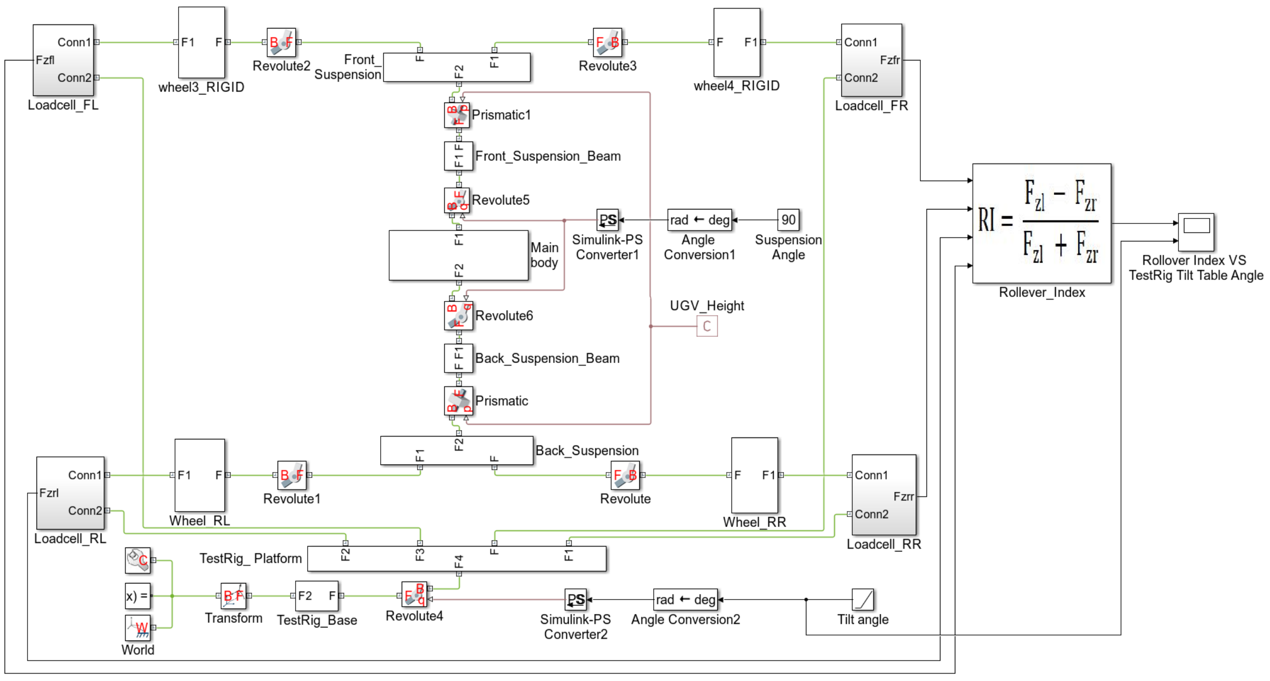


Fig. 4.2 Simulink UGV model on Test Rig

Test rig simulation

1. Simulation is set for the test rig to tilt from 0 to 85° over a period of 10 second
2. While vertical forces are measured on all four wheels
3. Then using the measured forces rollover index is computed using the following formula:

$$R = \frac{F_{zr} - F_{z\ell}}{F_{zr} + F_{z\ell}}$$

4. The rollover index is then plotted in Fig 4.4 with Fig 4.3 showing different stages of the simulation visualization.

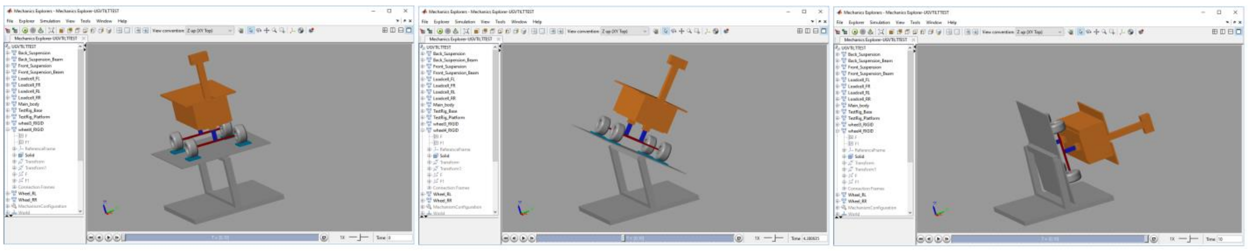
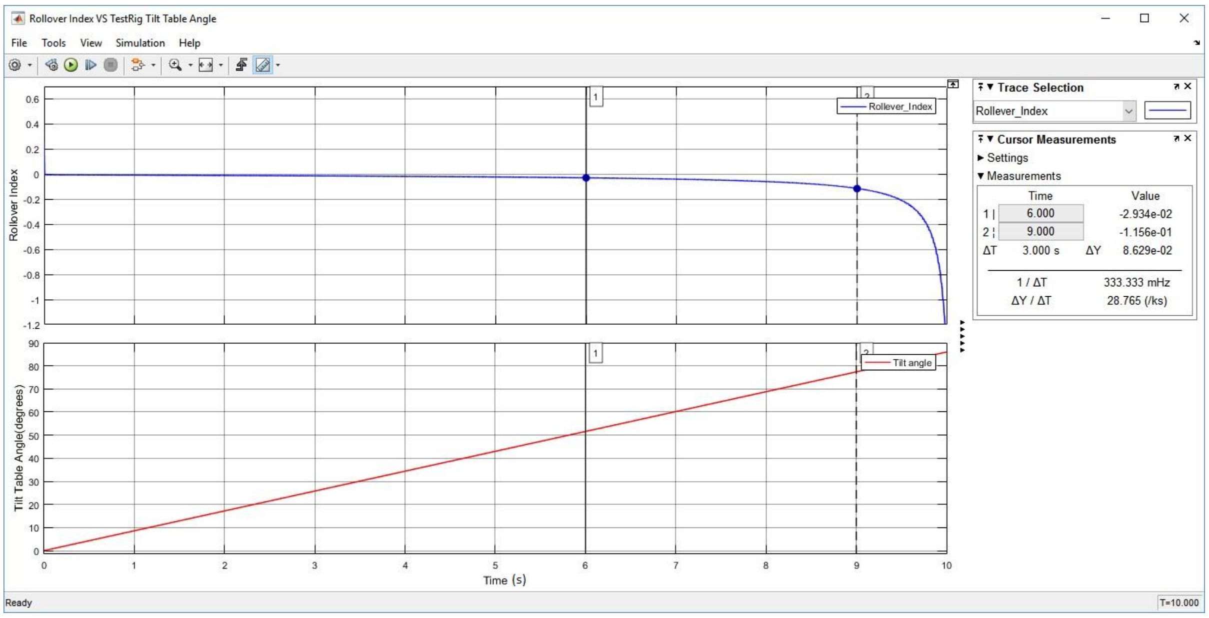


Fig. 4.3 Test Rig simulation visualisation

Fig. 4.4 shows the rollover index VS Tilt table angle over a 10 second period.



	Time		Value
1	6.000		-2.934e-02
2	9.000		-1.156e-01
ΔT	3.000 s	ΔY	8.629e-02
1 / ΔT			333.333 mHz
ΔY / ΔT			28.765 (/ks)

Fig 4.4 UGV Rollover Index VS Test Rig test table tilt angle without Adaptive Control

4.1.2 Test Rig simulation with adaptive controller.

The test setup is done as in section 4.1.1 but with adaptive controller implemented as per Fig. 4.5, with Fig. 4.6 showing the Rollover Index VS Test Rig test table tilt angle and Fig. 4.7 showing comparison between Rollover Index with (RI2) and Rollover Index without (RI1) Adaptive Control.

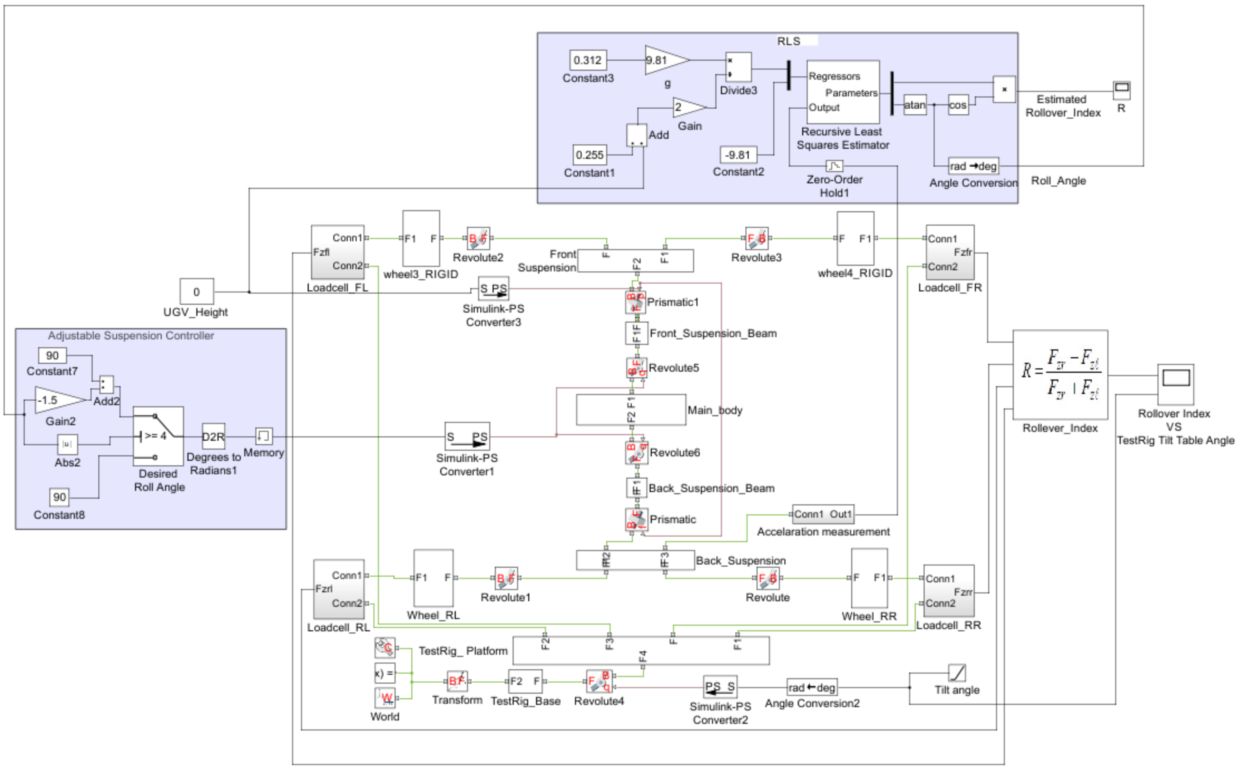
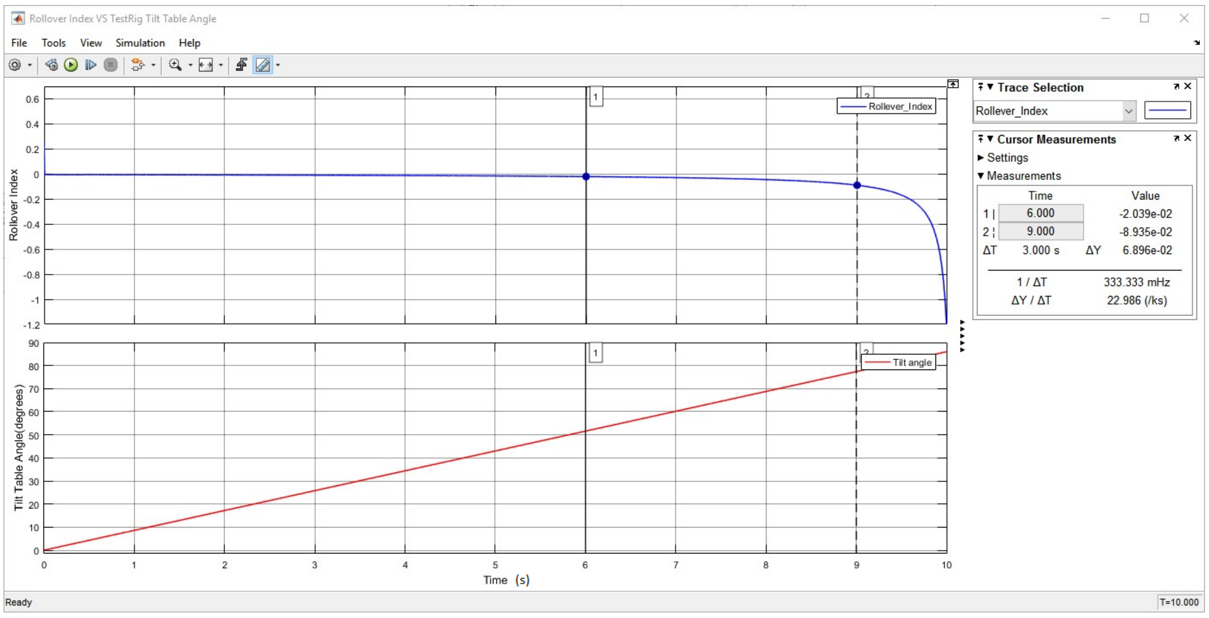


Fig 4.5 Simulink UGV model on Test Rig with Adaptive Controller



	Time		Value
1	6.000		-2.039e-02
2	9.000		-8.935e-02
ΔT	3.000 s	ΔY	6.896e-02
	1 / ΔT		333.333 mHz
	ΔY / ΔT		22.986 (/ks)

Fig. 4.6 UGV Rollover Index VS Test Rig test table tilt angle with Adaptive Control

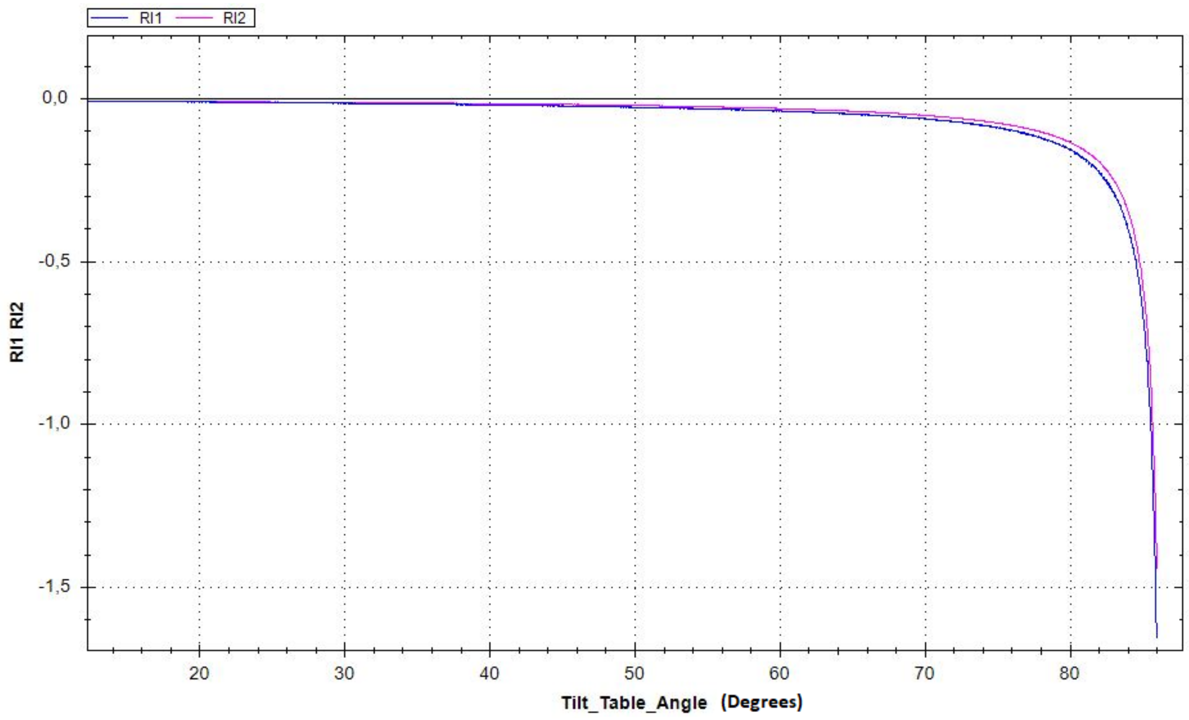


Fig. 4.7 UGV Rollover Index with (RI2) VS Rollover Index without Adaptive Control (RI1)

It can be seen on Fig 4.7 that rollover index of the UGV with adaptive control improved as compare to tilt table simulation of rollover index without adaptive control

4.2 UGV Multi-body Dynamic model Performance Evaluation

Dynamic model simulation will be conducted to analyse the applicability of adaptive control system in minimizing the vehicle rollover.

In this simulation the adaptive control system will be validated using a simulated UGV driving within a given trajectory and an animation model of the UGV is built in a virtual world using V-Realm Builder and is connected to the Simulink block using VR sink block.

Fig 4.8 shows the Simulink model used to simulate UGV driving through the given trajectory with the input and output parameters to the model listed in table 4.2

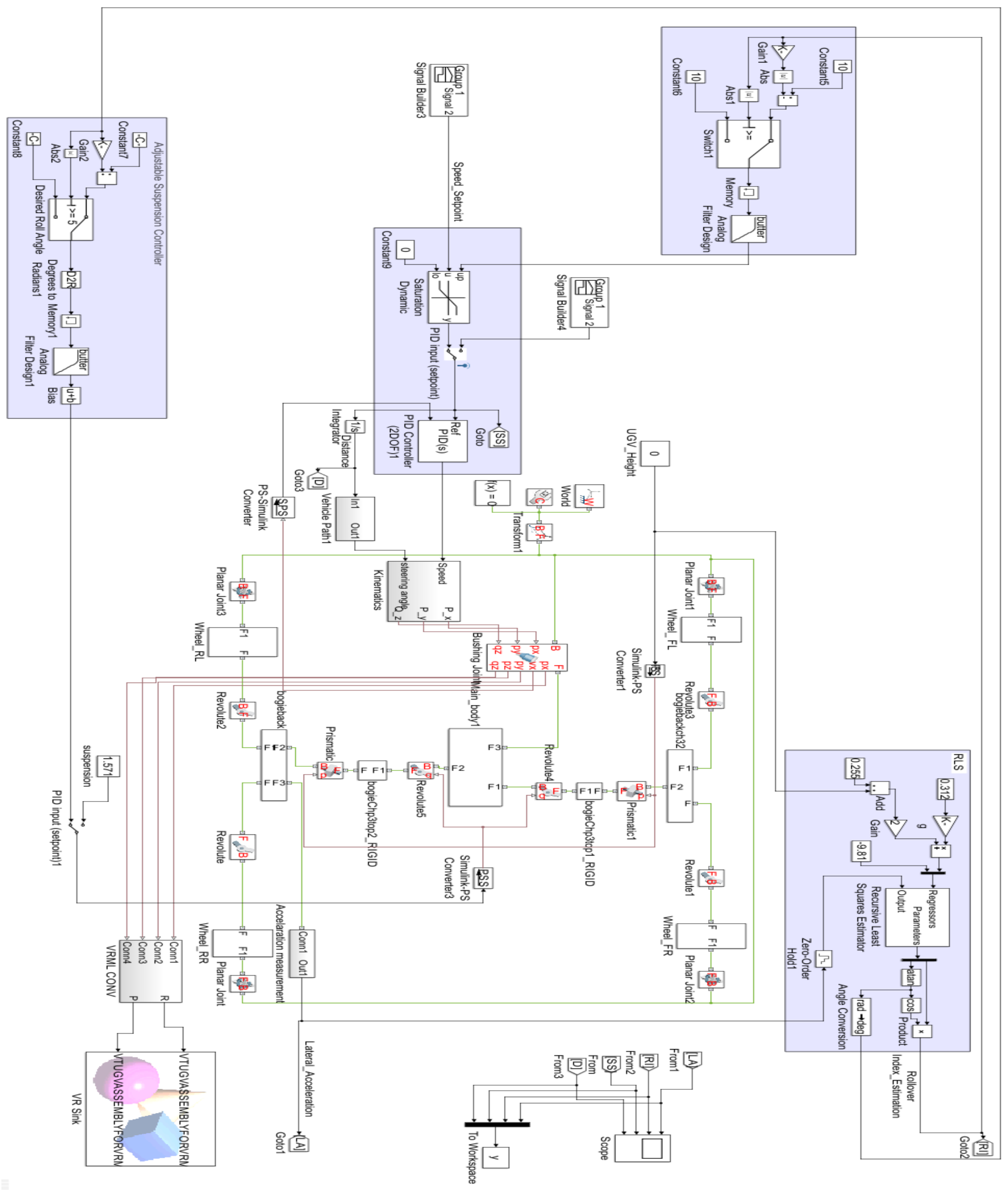


Fig. 4.8 UGV Simulink model

Table 4.2: UGV Model input and output parameters

INPUT		Description
1	Steering Angle	The vehicle is straight
2	Vehicle Speed	The vehicle is not moving
3	Suspension Angle	The is no deflection on the suspension
4	Height of centre of gravity	It's the input to the RLS Estimator
OUTPUT		**Description**
1	Lateral acceleration	Lateral acceleration measured at the back suspension un-sprung mass
2	Vehicle speed	Measured vehicle speed for PID speed Controller feedback

Adaptive control system can be switched on and off using switch 1 and the UGV is animated using matlab 3D animation.

The UGV simulation is setup as per Fig 4.8 with the vehicle speed set at 5m/s and it will start turning to the left on a curve at t > 2s, then at t > 8s the UGV will complete the curve and travels straight again following a curve in Fig 4.9

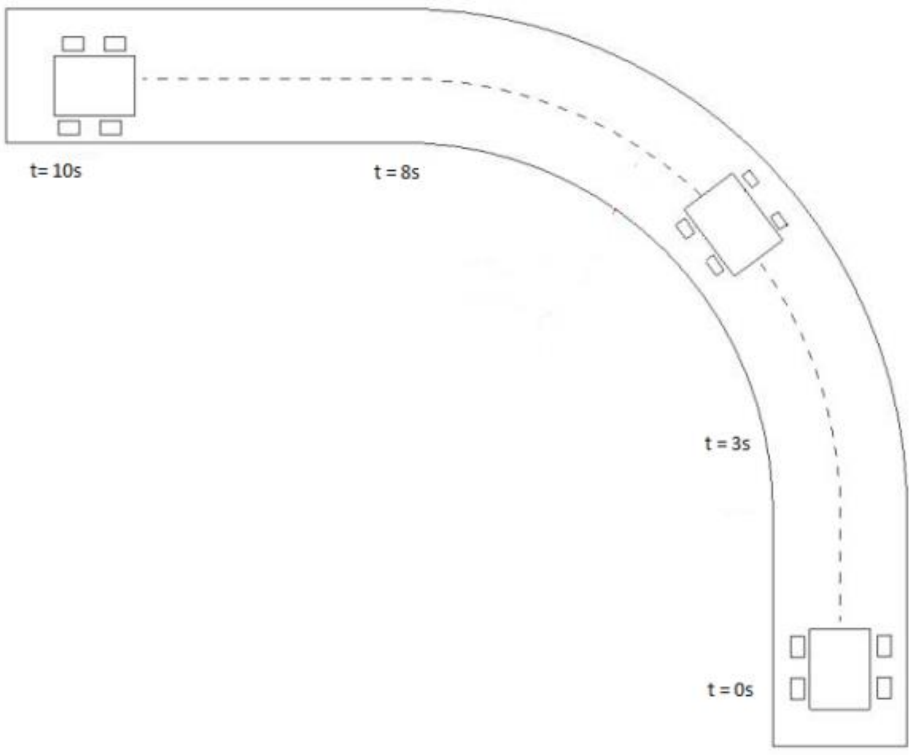


Fig. 4.9 UGV's path trajectory

4.2.1 Simulation without adaptive controller.

Firstly, the UGV is simulated without adaptive controller, negotiating a curve along the trajectory in Fig 4.9 with lateral acceleration measured at unsprung mass (vehicle suspension) and Fig 4.11 shows the results.

The following are the input parameters of the UGV model

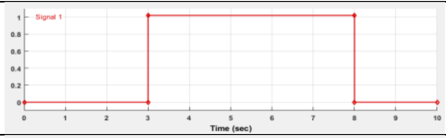
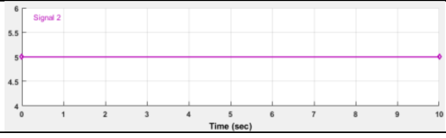
INPUT		Value	Description
1	Steering Angle.		The vehicle makes a constant steering angle of 1 degree from t>3 up to t =8
2	Vehicle Speed Command		Vehicle speed command set at 5m/s
3	Suspension Angle	0	The is no deflection on the suspension

Fig 4.10 shows the animated UGV in a virtual world as it travels along a given path

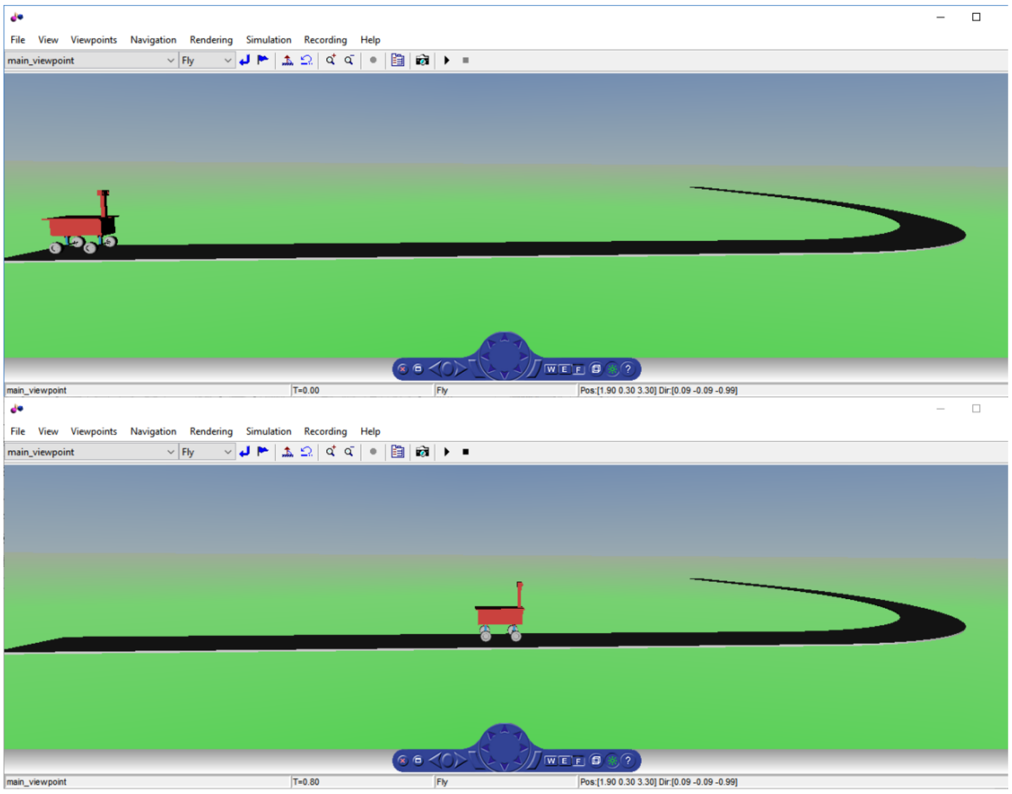
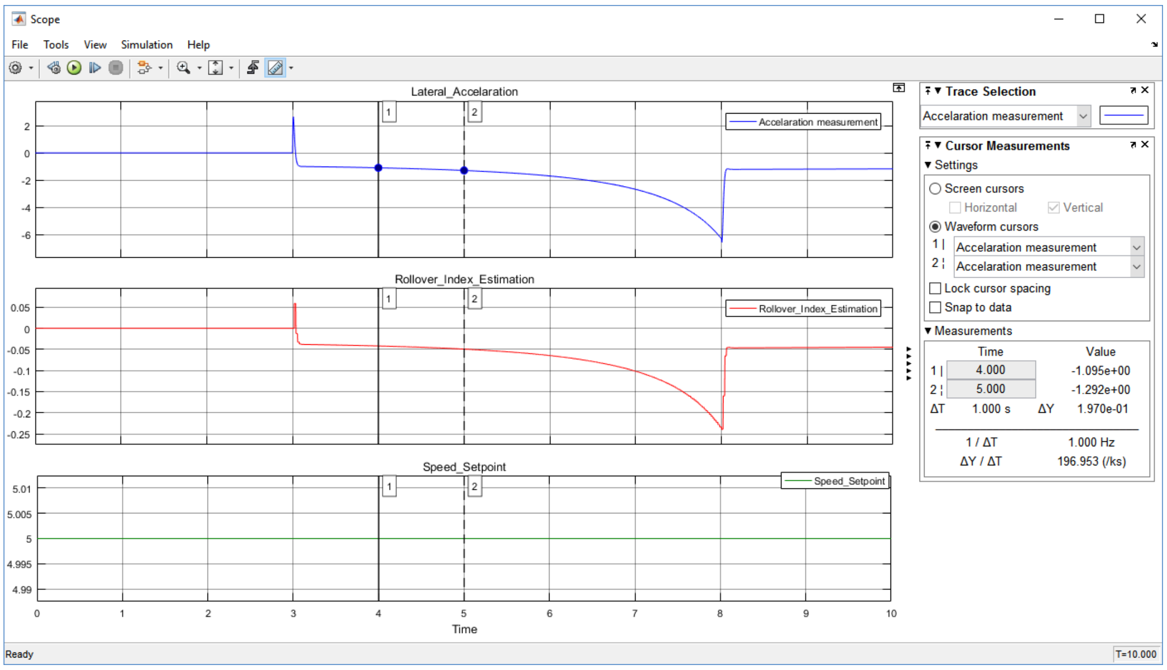

File View Viewpoints Navigation Rendering Simulation Recording Help
main_viewpoint
Fly
T=0.00
Pos:[1.90 0.30 3.30] Dir:[0.09 -0.09 -0.99]
T=0.80

Fig. 4.10 UGV on virtual world

The graphs on Fig 4.11 shows the lateral acceleration, rollover index estimation and speed-set point (which remains constant throughout the simulation)



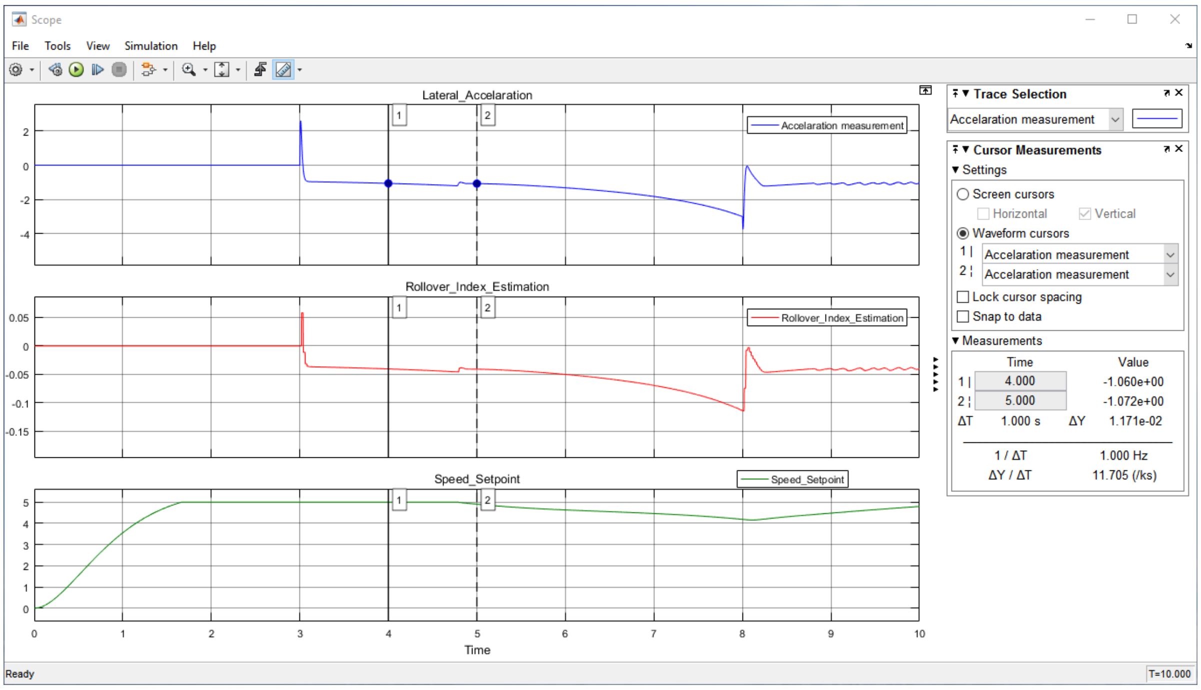
```
Trace Selection
Accelaration measurement
Cursor Measurements
Settings
( ) Screen cursors
  [ ] Horizontal   [✓] Vertical
(●) Waveform cursors
  1  Accelaration measurement
  2  Accelaration measurement
[ ] Lock cursor spacing
[ ] Snap to data
Measurements
        Time          Value
1      4.000        -1.095e+00
2      5.000        -1.292e+00
ΔT     1.000 s   ΔY  1.970e-01

   1 / ΔT          1.000 Hz
   ΔY / ΔT         196.953 (/ks)
Ready                                T=10.000
```

Fig. 4.11 UGV Lateral Acceleration VS Rollover Index Estimation VS Speed set point

4.2.2 Dynamic Model Simulation with adaptive controller.

Secondly the UGV is simulated with adaptive controller with the UGV negotiating a curve with lateral acceleration measured at unsprung mass (vehicle suspension) and Fig 4.12 shows the results.



```
Trace Selection
Accelaration measurement
Cursor Measurements
Settings
( ) Screen cursors
  [ ] Horizontal   [✓] Vertical
(●) Waveform cursors
  1  Accelaration measurement
  2  Accelaration measurement
[ ] Lock cursor spacing
[ ] Snap to data
Measurements
        Time          Value
1      4.000        -1.060e+00
2      5.000        -1.072e+00
ΔT     1.000 s   ΔY  1.171e-02

   1 / ΔT          1.000 Hz
   ΔY / ΔT         11.705 (/ks)
Ready                                T=10.000
```

Fig. 4.12 UGV Lateral Acceleration VS Rollover Index Estimation VS Speed set point

Taking a snap shot of the two graphs at t=4s and t = 5s it points out that lateral acceleration lowers if adaptive control is implemented on the UGV control system and also the speed set-point is automatically reduced without varying the speed command.

Chapter 5 Analysis and discussion of Results

In this chapter analysis of the simulated results between a UGV with adaptive controller (both adaptive speed control and adaptive suspension control) and the one without adaptive controller is presented.

5.1 Simulation Setup

In order to analyse the performance of the proposed adaptive control system algorithm the UGV Model in chapter 3 Fig 4.8 is simulated traveling along an S- curve trajectory and a J- curve trajectory in Fig 5.1 and Fig 5.2 respectively using Matlab Simulink simulation software.

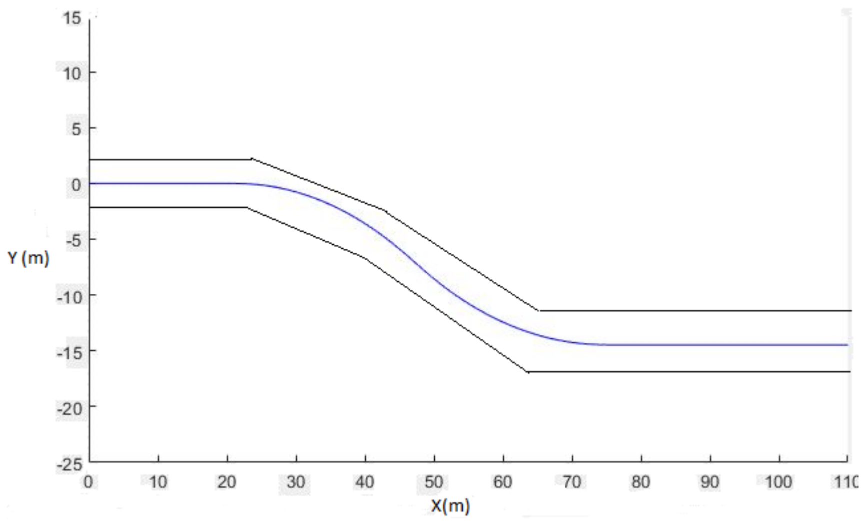


Fig. 5.1 S- curve trajectory

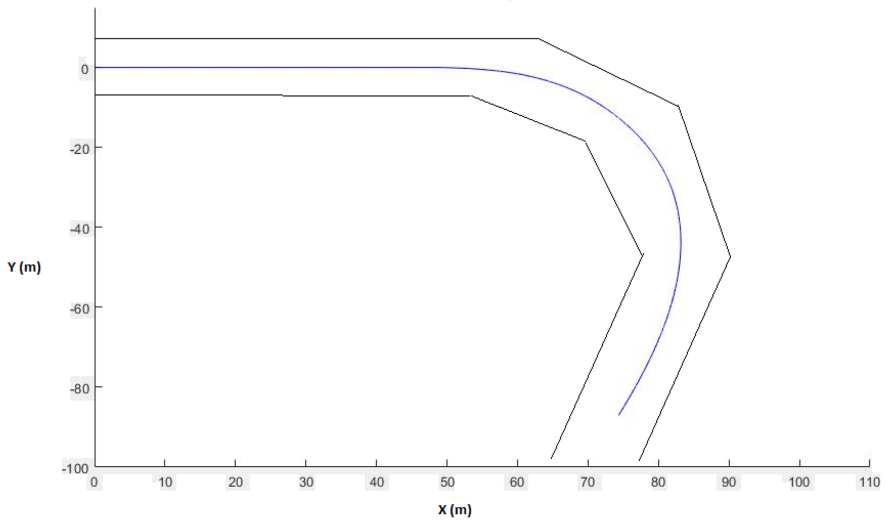


Fig. 5.2 J- curve trajectory

5.2 Simulation Results

5.2.1 UGV travelling through an S- curve trajectory

Fig 5.3-5.5 showing comparison of UGV's lateral acceleration, rollover index and speed (for both without adaptive controller $a_{y1}$and the one with adaptive controller a_{y2}) for the UGV travelling through an S –curve trajectory.

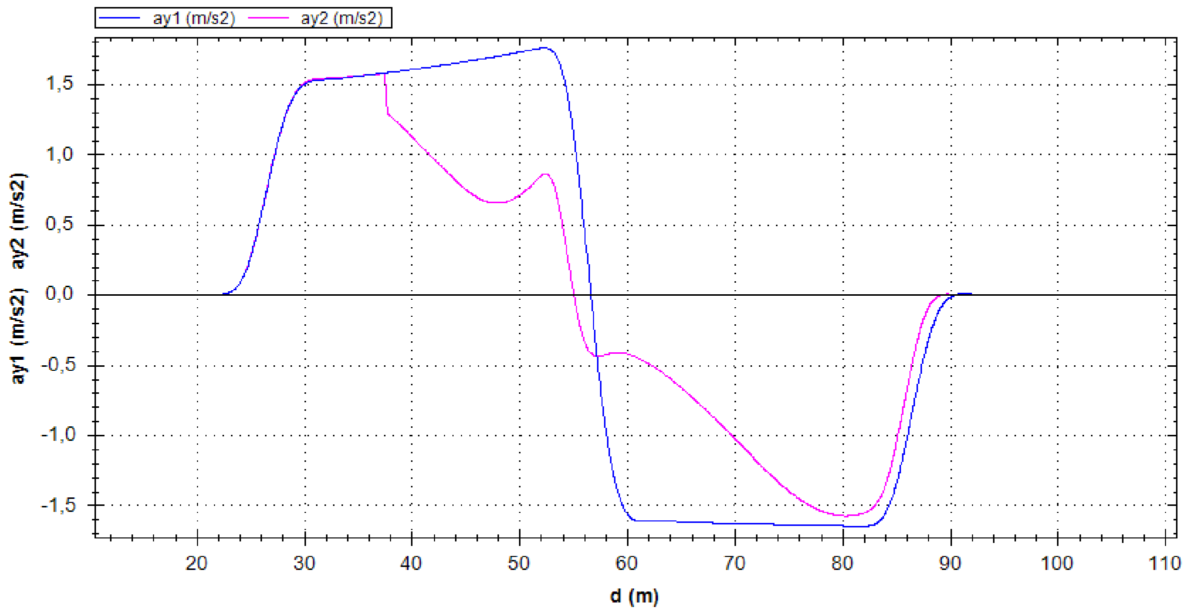


Fig. 5.3 Lateral acceleration simulation results: a_{y1} vs a_{y2} against distance travelled

Looking at Fig 5.3 between 30 and 40 metres when the UGV is negotiating a right turn curve the lateral acceleration (a_{y1}) of the UGV without adaptive controller continues to increase beyond $1.5m/s^2$ to maximum of $1.7m/s^2$ until the vehicle passes the curve and start negotiating a left turn curve which again lateral acceleration of the UGV without adaptive controller increases to maximum of $1.6m/s^2$.

But the UGV with adaptive control at around 37m lateral acceleration (a_{y2}) decreases while UGV is still negotiating a curve comparing it to UGV without adaptive controller at 52.5m where maximum lateral acceleration: $a_{y1} = 1.7m/s^2$ while $a_{y2} = 0.8m/s^2$

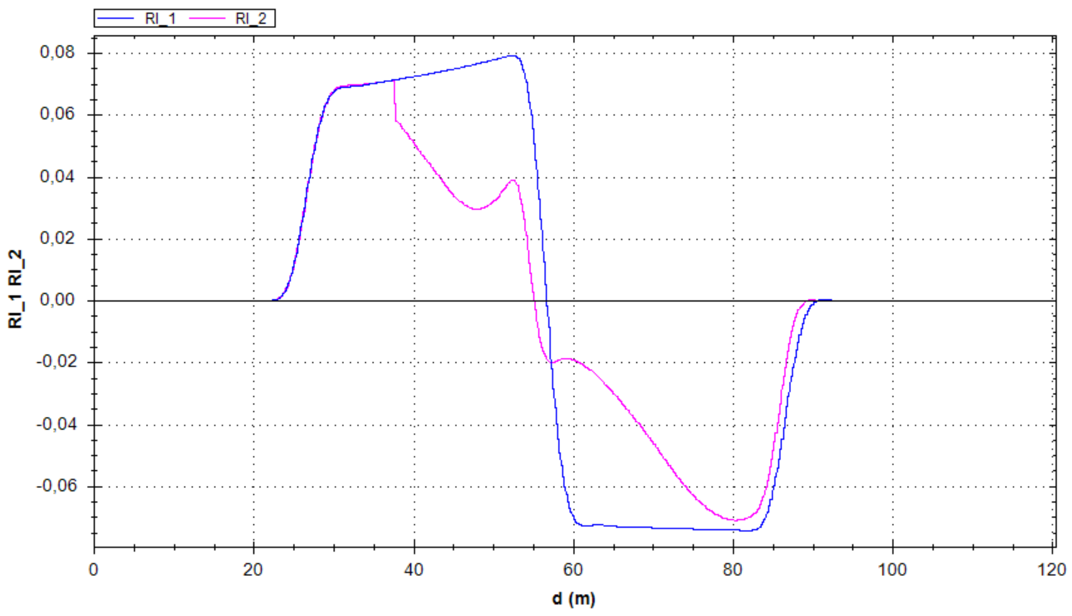


Fig. 5.4 Rollover index simulation results RI1 vs RI2 against distance travelled

Looking at Fig 5.4 compering the estimated rollover index of the UGV without adaptive control (RI_1) and the one with adaptive control (RI_2) at around 37m the Roll over Index (RI_2) decreases while UGV is still negotiating a curve comparing it to UGV without adaptive controller at 52.5m where Rollover Index: RI_1 = 0.079 while RI_2 = 0.0389.

Also when the UGV is negotiating a left turn on the S-curve at around 80m Rollover Index: RI_1 = 0.074 while RI_2 = 0.071.

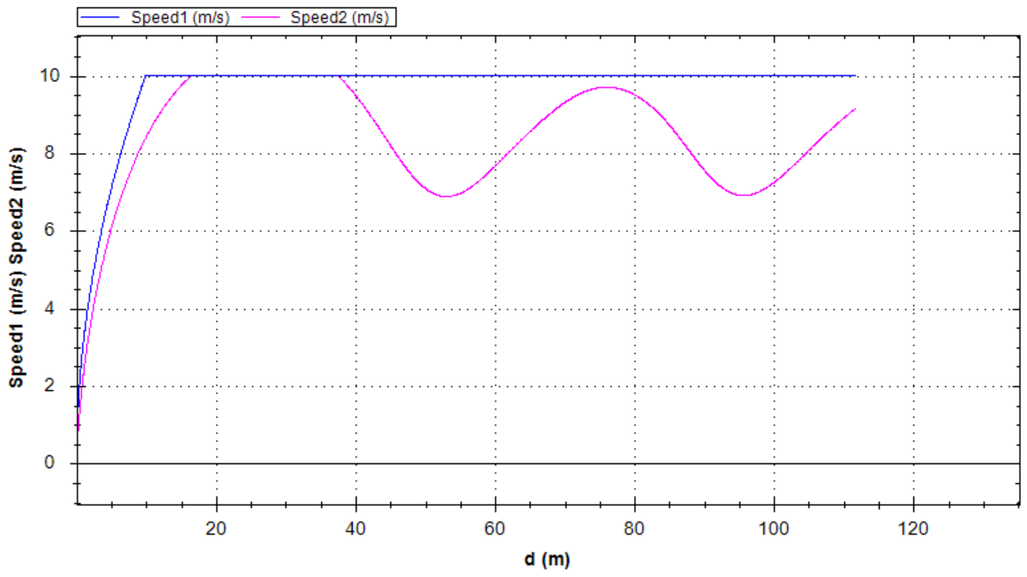


Fig. 5.5 Speed command simulation results

It can be seen on fig 5.6 that the UGV speed (Speed2) for the UGV with adaptive control between 35 and 50 metres decreases resulting on the decrease of lateral acceleration as seen on fig 5.5 and subsequence reduction in rollover index to below the 0.04 desired threshold (Desired UGV performance) while the UGV without adaptive control increases to 0.08 which is 0.04 above the desired performance.

From this simulation outcome it can be deduced that the adaptive control system does lower the prosperity for vehicle rollover thereby increasing the vehicles reliability and survivability while negotiating an S-curve trajectory

5.2.2 UGV travelling through a J- curve trajectory

Fig. 5.7-5.9 showing comparison of UGV's lateral acceleration, rollover index and speed (for both without adaptive controller $a_{y1}$and the one with adaptive controller a_{y2}) for the UGV travelling through a J –curve trajectory.

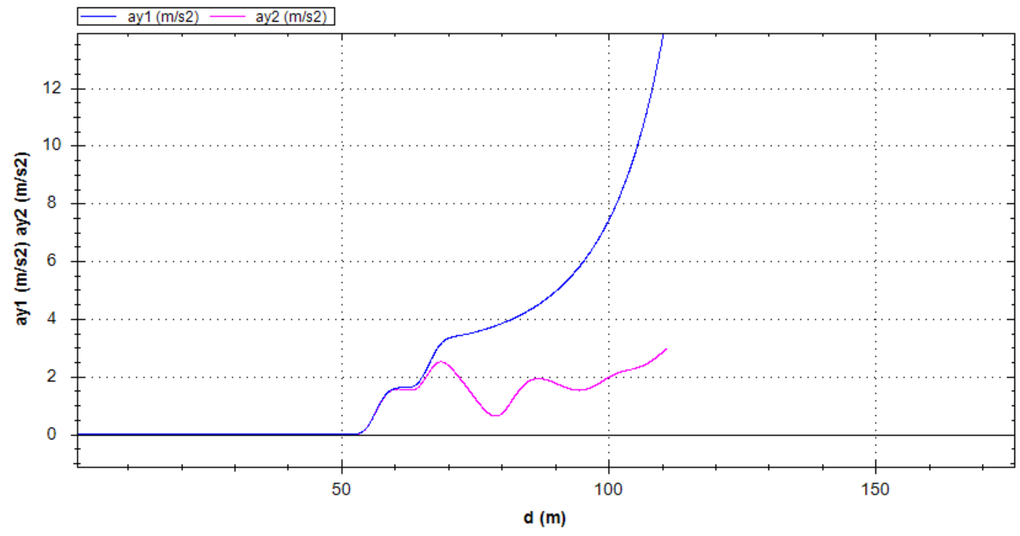


Fig. 5.6 Lateral acceleration simulation results: a_{y1} vs a_{y2} against distance travelled

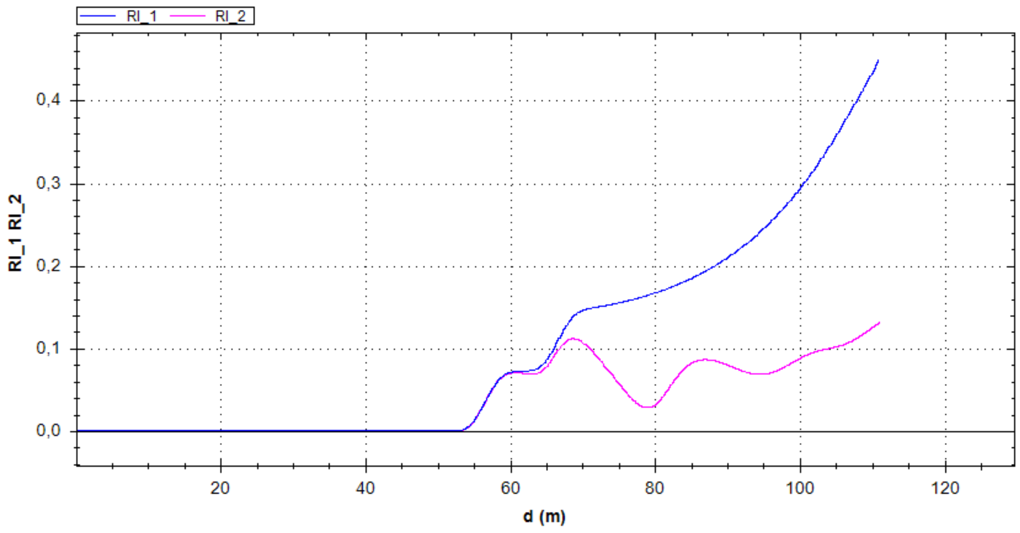


Fig. 5.7 Rollover index simulation results RI1 vs RI2 against distance travelled

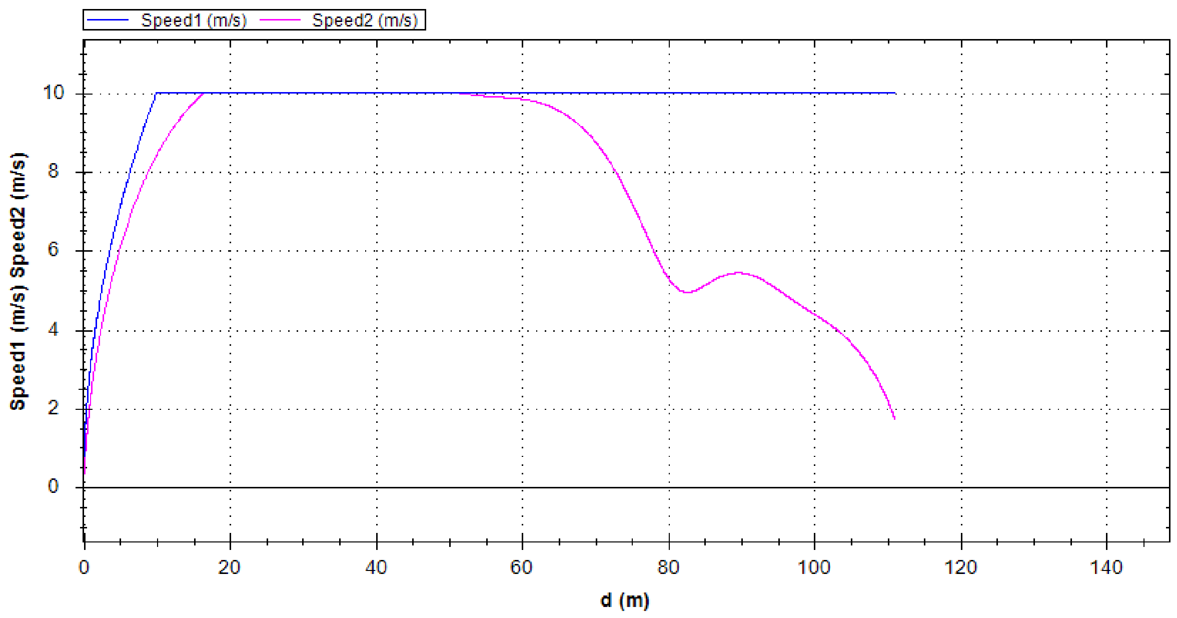


Fig. 5.8 Speed command simulation results

Looking at Fig 5.6 and Fig 5.7 the UGV start negotiating a J –Curve at about 55m and the lateral acceleration start increasing so as the rollover index but it can be seen that on the UGV without adaptive control the lateral acceleration and rollover Index continues to increase as the UGV negotiate the J-Curve but for the UGV with adaptive control at about 70m the rollover index and lateral acceleration start decreasing as the speed decreases.

From this simulation outcome it can be deduced that the adaptive control system does lower the prosperity for vehicle rollover thereby increasing the vehicles reliability and survivability while negotiating a J-curve trajectory.

Therefore, both simulation results show that adaptive control system based on RLS estimation of the roll angle and rollover index does prevent vehicle rollover in real time.

Chapter 6 Conclusion

6.1 Conclusion

Determining the rollover index in real time is paramount to any vehicle rollover preventions system therefore this paper focuses on development of real time rollover index estimation algorithms in order to activate in real time the rollover prevention systems (active suspension system and speed control system).

The developed adaptive control system algorithm based on real time recursive least squares estimation of the rollover index and Roll angle has been evaluated using Matlab/Simulink simulations and the simulation results confirmed that the adaptive control algorithms does prevent or minimize the likelihood of vehicle rollover thereby increasing the UGV's reliability, survivability and adaptability on unknown terrains and it can be cost effectively implemented on an open source Arduino board programmed using matlab Simulink as described by Adaptive Control System Simulink model in appendix A and the C code for Arduino board in appendix B.

This research also presents a way of roll angle estimation without the use of expensive sensors, electronic hardware and high computation requirement.

6.2 Future work

This research dealt with only simulations of the UGV to prove the applicability of adaptive control technique in mitigating the vehicle rollover; therefore, further research needs to be conducted on the real UGV.

LIST OF PUBLICATIONS

1. Mlati, M.C., Wang, Z: Unmanned Ground vehicles: Adaptive Control System for Real-Time Rollover Prevention International Journal of Vehicle Autonomous Systems.

AWARDS

1. Mlati, M.C., Wang, Z: Unmanned Ground vehicles: Adaptive Control System for Real-Time: was selected as a 1st Runner up: Natural and Physical Science (Master Category) for long paper on the 7th Annual Unisa Student Research and Innovation Showcase.

Bibliography

[1] National Aeronautics and Space Administration (2004): Mars Exploration Rover Factsheet. Retrieved from: http://mars.nasa.gov/mer/newsroom/factsheets/pdfs/Mars03Rover041020.pdf .

[2] Clancey, W.J. (2012). Working on Mars:Voyages of Scientific Discovery with the Mars Exploration Rovers; Cambridge: MIT Press.

[3] Nguyen-Huu, P., Titus, J., Tilbury, D. & Ulsoy, A.G., (2009) Reliability and Failure in Unmanned Ground Vehicle (UGV).
Retrieved from:
http://arc.engin.umich.edu/grrc/techreports/200901_ReliabilityUGV.pdf

[4] Landau, I.D., Lozano, R., M'Saad, M. & Karimi, A. (2011) Adaptive Control Algorithms, Analysis and Applications. London ; New York : Springer

[5] Spenko, M., Overholt, J. & Lagnemma, K. (2006) High Speed Hazard Avoidance for Unmanned Ground Vehicles in Emergency Situations.
Retrieved from:
http://web.mit.edu/mobility/publications/Iagnemma_ASC_06.pdf

[6] Alajlan, A.M., Almasri, M.M. & Elleithy, K.M. (2015). Multi-Sensor Based Collision Avoidance Algorithm for Mobile Robot. *IEEE Long Island Systems, Applications and Technology Conference*. 1-6.

[7] Liu, J., Wang Y., Bin, L.I., & Shugen, M.A. (2006). Current research, key performances and future development of search and rescue robots. *Chinese Journal of Mechanical Engineering*. 42(12), 1–12.

[8] Rajamani, R. (2012) .Vehicle Dynamics and Control. Mechanical Engineering Series. Springer, Boston, MA

[9] Kazemian, A.H., Fooladi, M. & Darijani, H. (2017) Rollover Index for the Diagnosis of Tripped and Untripped Rollovers. *Latin American Journal of Solids and Structures*, 14(11), 1979-1999

[10] Rajamani, R., Piyabongkarn, D., Tsourapas,V. & Lew, J.Y. (2011) Parameter and State Estimation in Vehicle Roll Dynamics. *IEEE Transactions on Intelligent Transportation Systems*. 12(4), 1558-1567.

[11] Dai, L., Xia, Y., Fu, M. & Mahmoud, M.S. (2012) Discrete-Time Model Predictive Control. Advances in Discrete Time Systems, Book Series, Intech.

[12] Fukushima, H., Kim,T.-H. & Sugie, T. (2007) Adaptive model predictive control for a class of constrained linear systems based on the comparison model. *Automatica Journal.* 43(2), 301-308.

[13] Krenn, R., Gibbesch, A., Binet, G., & Bemporad, A. (2013) Model Predictive Traction and Steering Control of Planetary Rovers. Retrieved from: http://robotics.estec.esa.int/ASTRA/Astra2013/Papers/Krenn_2811412.pdf.

[14] Mahmoud, M.S. & Xia, Y. (2012) Applied Control systems Design. New York: Springer.

[15] Eberhardt, R.L. & Ward, D.G. (1999) Indirect Adaptive Flight Control System Interactions. *International Journal of Robust and Nonlinear Control.* 9, 1013-1031.

[16] Jiang, J. & Zhang, Y. (2004) A revisit to block and recursive least squares for parameter estimation. *Computers and Electrical Engineering.* 30 403–416.

[17] Keesman, K.J. (2011).System Identification. Springer-Verlag London.

[18] Young, P.C. (2011) Recursive Estimation and Time-Series Analysis; Springer Heidelberg Dordrecht London New York.

[19] Schofield, B. (2006) Vehicle Dynamics Control for Rollover Prevention Department of Automatic Control, Lund Institute of Technology, Lund University.

[20] Kim, D., Choi, S. B., & Oh, J. (2012) Integrated Vehicle Mass Estimation using Longitudinal and Roll Dynamics. *International Conference on Control, Automation and Systems*. 12, 862-867.

[21] Chowdhary G., Mühlegg M., How J.P., Holzapfel F. (2013) Concurrent Learning Adaptive Model Predictive Control. *Advances in Aerospace Guidance, Navigation and Control.* 29-47

[22] Carlson, J., Murphy, R. R., & Nelson, A. (2004) Follow-up Analysis of Mobile Robot Failures*", Robotics and Automation 2004. Proceedings. IEEE International Conference.* 5, 4987-4994.

[23] Holkar, K. S., & Waghmare, L. M. (2010). An Overview of Model Predictive Control. *International Journal of Control and Automation*. 3(4), 47-64.

[24] Prabhu, K., & Bhaskaran, V.M. (2012) Optimization of a Control Loop Using Adaptive Method. *International Journal of Engineering and Innovative Technology (IJEIT).* 1(3), 133-138

[25] Grüne, L., & Pannek, J. (2011). Nonlinear Model Predictive Control: Theory and Algorithm. Springer-Verlag London Limited.

[26] Spenko, M., Kuroda, Y., Dubowsky, S. & Iagnemma, K. (2006) Hazard avoidance for high-speed mobile robots in rough terrain. J. Field Robotics. 23, 311–331.

[27] Predictive Control (2013). Retrieved from: http://en.wikipedia.org/wiki/Model_predictive_control.

[28] Sivaji, V.V., & Sailaja, M. (2013) Adaptive Cruise Control Systems for Vehicle Modeling Using Stop and Go Manoeuvres. International Journal of Engineering Research and Applications (IJERA), 3(4), 2453-2456

[29] Adaptive Control (2013). Retrieved from: http://en.wikipedia.org/wiki/Adaptive_control

[30] Mohamed, Y.S., Hasaneen, B. M., Elbaset, A.A., & Hussein, A.E. (2010) Recursive Least Square Algorithm for Estimating Parameters Of An Induction Motor. Journal of Engineering Sciences, *Assiut University*, 39(1), 87-98

[31] Binet G., Krenn,R., & Bemporad A. (2012) Model predictive control applications for planetary rovers. *i-SAIRAS 2012*.

[32] Oh, K.S., & Seo, J.H. (2017) An adaptive position control algorithm of a DC motor based on the first order approximation using recursive least squares with forgetting. *International Conference on Control, Automation and Systems (ICCAS)*. 17, 1858-1861.

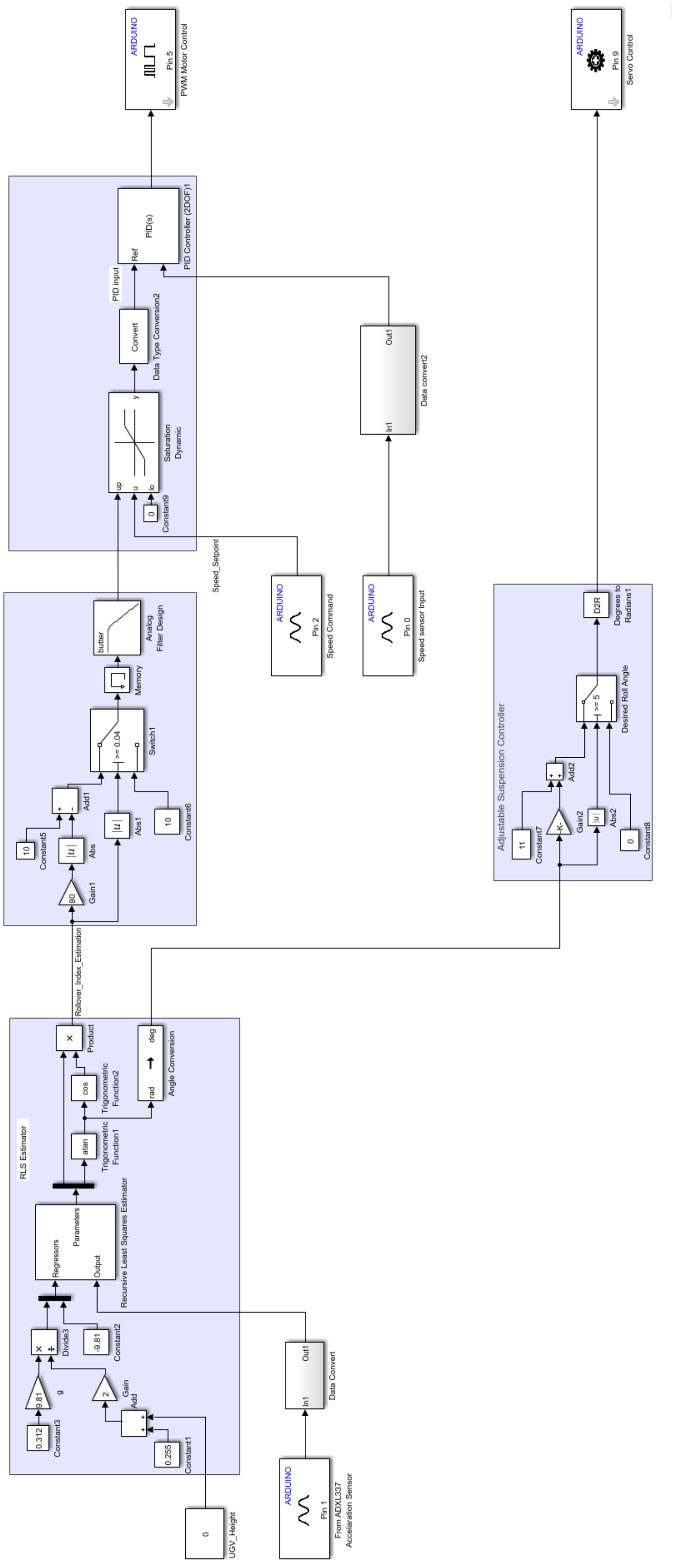
[33] Lee, S., Yakub, F., Kasahara, M., & Mori, Y. (2013) Rollover prevention with Predictive Control of differential braking and rear wheel steering. *IEEE Conference on Robotics, Automation and Mechatronics (RAM)*. 6, 144-149.

[34] Richier, M., Lenain, R., Thuilot, B., & Debain, C. (2013) Rollover prevention of All-Terrain Vehicle during aggressive driving using multi-model observer. Application to ATVS in off-road context. *IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT).*

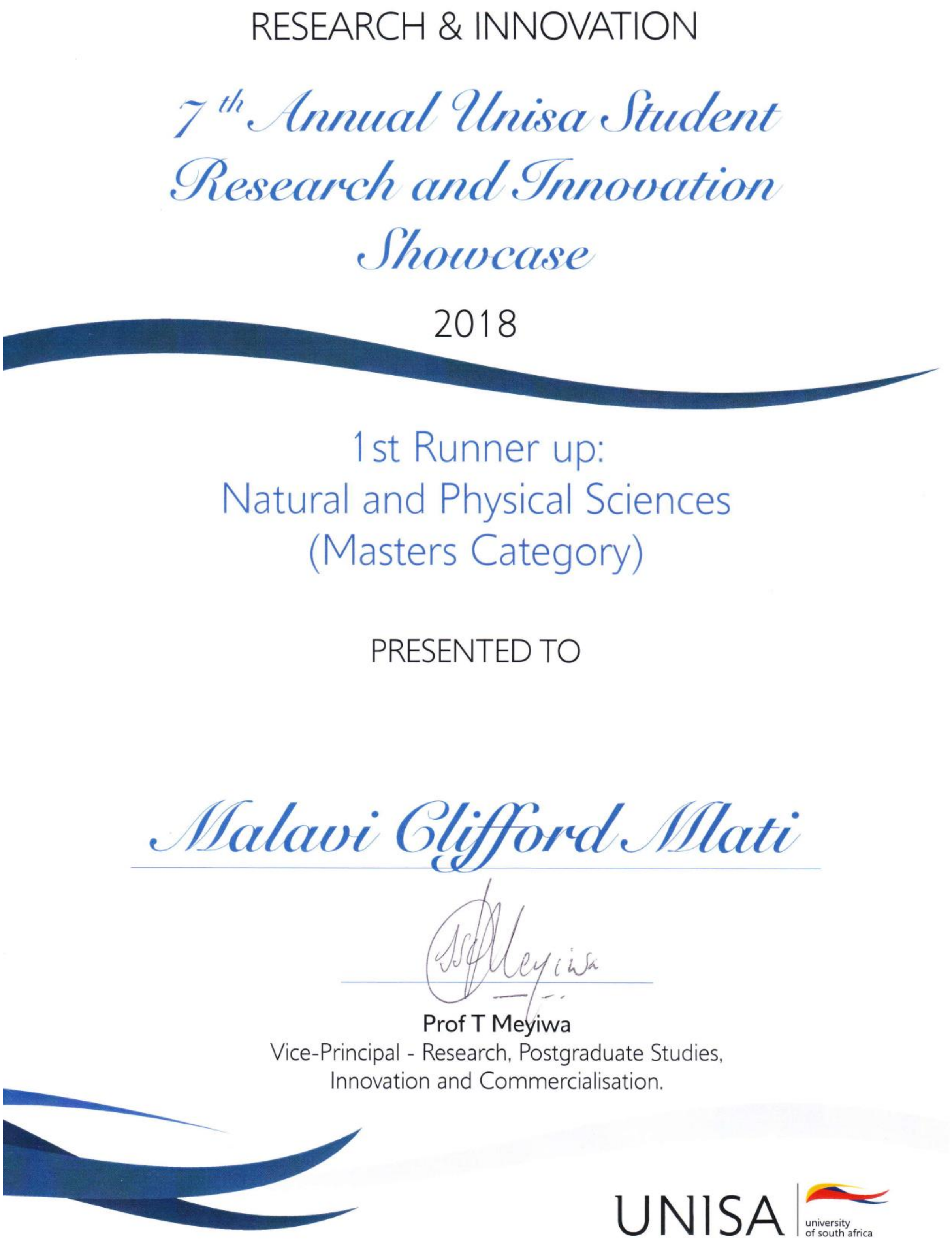
[35] Stoica, P. and Ahgren P., (2002) Exact initialization of the recursive least squares algorithm. *International Journal of Adaptive Control and Signal Processing,* vol. 16, pp. 219-230

APPENDICES

Appendix A: UGV Adaptive Control System Simulink model for Arduino board.



Appendix B: 7th Annual Unisa Student Research and Innovation Showcase research award



RESEARCH & INNOVATION

7th Annual Unisa Student Research and Innovation Showcase

2018

1st Runner up:
Natural and Physical Sciences
(Masters Category)

PRESENTED TO

Malavi Clifford Mlati

[illegible]

Prof T Meyiwa
Vice-Principal - Research, Postgraduate Studies,
Innovation and Commercialisation.

UNISA university of south africa

Appendix C: Main UGV C code for Arduino board

```
 /*
  * UGV_Arduino_code.c
  *
  * Student License - for use by students to meet course requirements and
  * perform academic research at degree granting institutions only.  Not
  * for government, commercial, or other organizational use.
  *
  * Code generation for model "UGV_Arduino_code".
  *
  * Model version              : 1.10
  * Simulink Coder version : 8.11 (R2016b) 25-Aug-2016
  * C source code generated on : Mon Mar 19 22:15:44 2018
  *
  * Target selection: ert.tlc
  * Note: GRT includes extra infrastructure and instrumentation for prototyping
  * Embedded hardware selection: Atmel->AVR
  * Code generation objectives: Unspecified
  * Validation result: Not run
  */

#include "UGV_Arduino_code.h"
#include "UGV_Arduino_code_private.h"

/* Block signals (auto storage) */
B_UGV_Arduino_code_T UGV_Arduino_code_B;

/* Continuous states */
X_UGV_Arduino_code_T UGV_Arduino_code_X;

/* Block states (auto storage) */
DW_UGV_Arduino_code_T UGV_Arduino_code_DW;

/* Previous zero-crossings (trigger) states */
PrevZCX_UGV_Arduino_code_T UGV_Arduino_code_PrevZCX;

/* Real-time model */
RT_MODEL_UGV_Arduino_code_T UGV_Arduino_code_M_;
RT_MODEL_UGV_Arduino_code_T *const UGV_Arduino_code_M = &UGV_Arduino_code_M_;

/* This function updates continuous states using the ODE8 fixed-step
 * solver algorithm
 */
static void rt_ertODEUpdateContinuousStates(RTWSolverInfo *si )
{
  /* Solver Matrices */
#define UGV_Arduino_code_NSTAGES       13

  static real_T rt_ODE8_B[13] = {
    4.174749114153025E-2, 0.0, 0.0, 0.0,
    0.0, -5.54523286112393E-2, 2.393128072011801E-1, 7.03510669403443E-1,
    -7.597596138144609E-1, 6.605630309222863E-1, 1.581874825101233E-1,
    -2.381095387528628E-1, 2.5E-1
  };

  static real_T rt_ODE8_C[13] = {
    0.0, 5.555555555555556E-2, 8.333333333333333E-2, 1.25E-1,
    3.125E-1, 3.75E-1, 1.475E-1, 4.65E-1,
    5.648654513822596E-1, 6.5E-1, 9.246562776405044E-1, 1.0, 1.0
  };

  static real_T rt_ODE8_A[13][13] = {
    /* rt_ODE8_A[0][] */
```

```
{ 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0 },

/* rt_ODE8_A[1][] */
{ 5.555555555555556E-2, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0 },

/* rt_ODE8_A[2][] */
{ 2.083333333333333E-2, 6.25E-2, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0 },

/* rt_ODE8_A[3][] */
{ 3.125E-2, 0.0, 9.375E-2, 0.0,
  0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0 },

/* rt_ODE8_A[4][] */
{ 3.125E-1, 0.0, -1.171875, 1.171875,
  0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0 },

/* rt_ODE8_A[5][] */
{ 3.75E-2, 0.0, 0.0, 1.875E-1,
  1.5E-1, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0 },

/* rt_ODE8_A[6][] */
{ 4.791013711111111E-2, 0.0, 0.0, 1.122487127777778E-1,
  -2.550567377777778E-2, 1.284682388888889E-2, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0 },

/* rt_ODE8_A[7][] */
{ 1.691798978729228E-2, 0.0, 0.0, 3.878482784860432E-1,
  3.597736985150033E-2, 1.969702142156661E-1, -1.727138523405018E-1, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0 },

/* rt_ODE8_A[8][] */
{ 6.90957533591923E-2, 0.0, 0.0, -6.342479767288542E-1,
  -1.611975752246041E-1, 1.386503094588253E-1, 9.409286140357563E-1,
  2.11636326481944E-1,
  0.0, 0.0, 0.0, 0.0, 0.0 },

/* rt_ODE8_A[9][] */
{ 1.835569968390454E-1, 0.0, 0.0, -2.468768084315592,
  -2.912868878163005E-1, -2.647302023311738E-2, 2.8478387641928,
  2.813873314698498E-1,
  1.237448998633147E-1, 0.0, 0.0, 0.0, 0.0 },

/* rt_ODE8_A[10][] */
{ -1.215424817395888, 0.0, 0.0, 1.667260866594577E1,
  9.157418284168179E-1, -6.056605804357471, -1.600357359415618E1,
  1.484930308629766E1,
  -1.337157573528985E1, 5.134182648179638, 0.0, 0.0, 0.0 },

/* rt_ODE8_A[11][] */
{ 2.588609164382643E-1, 0.0, 0.0, -4.774485785489205,
  -4.350930137770325E-1, -3.049483332072241, 5.577920039936099,
  6.155831589861039,
  -5.062104586736938, 2.193926173180679, 1.346279986593349E-1, 0.0, 0.0 },

/* rt_ODE8_A[12][] */
```

```
  { 8.224275996265075E-1, 0.0, 0.0, -1.165867325727766E1,
    -7.576221166909362E-1, 7.139735881595818E-1, 1.207577498689006E1,
    -2.127659113920403,
    1.990166207048956, -2.342864715440405E-1, 1.758985777079423E-1, 0.0, 0.0 },
};

time_T t = rtsiGetT(si);
time_T tnew = rtsiGetSolverStopTime(si);
time_T h = rtsiGetStepSize(si);
real_T *x = rtsiGetContStates(si);
ODE8_IntgData *intgData = (ODE8_IntgData *) rtsiGetSolverData(si);
real_T *deltaY = intgData->deltaY;
real_T *x0 = intgData->x0;
real_T* f[UGV_Arduino_code_NSTAGES];
int idx,stagesIdx,statesIdx;
double deltaX;
int_T nXc = 5;
rtsiSetSimTimeStep(si,MINOR_TIME_STEP);
f[0] = intgData->f[0];
f[1] = intgData->f[1];
f[2] = intgData->f[2];
f[3] = intgData->f[3];
f[4] = intgData->f[4];
f[5] = intgData->f[5];
f[6] = intgData->f[6];
f[7] = intgData->f[7];
f[8] = intgData->f[8];
f[9] = intgData->f[9];
f[10] = intgData->f[10];
f[11] = intgData->f[11];
f[12] = intgData->f[12];

/* Save the state values at time t in y and x0*/
(void) memset(deltaY, 0,
              (uint_T)nXc*sizeof(real_T));
(void) memcpy(x0, x,
              nXc*sizeof(real_T));
for (stagesIdx=0;stagesIdx<UGV_Arduino_code_NSTAGES;stagesIdx++) {
  (void) memcpy(x, x0,
                (uint_T)nXc*sizeof(real_T));
  for (statesIdx=0;statesIdx<nXc;statesIdx++) {
    deltaX = 0;
    for (idx=0;idx<stagesIdx;idx++) {
      deltaX = deltaX + h*rt_ODE8_A[stagesIdx][idx]*f[idx][statesIdx];
    }

    x[statesIdx] = x0[statesIdx] + deltaX;
  }

  if (stagesIdx==0) {
    rtsiSetdX(si, f[stagesIdx]);
    UGV_Arduino_code_derivatives();
  } else {
    (stagesIdx==UGV_Arduino_code_NSTAGES-1)? rtsiSetT(si, tnew) : rtsiSetT(si,
      t + h*rt_ODE8_C[stagesIdx]);
    rtsiSetdX(si, f[stagesIdx]);
    UGV_Arduino_code_step();
    UGV_Arduino_code_derivatives();
  }

  for (statesIdx=0;statesIdx<nXc;statesIdx++) {
    deltaY[statesIdx] = deltaY[statesIdx] + h*rt_ODE8_B[stagesIdx]*f[stagesIdx]
      [statesIdx];
  }
```

```
    }

    for (statesIdx=0;statesIdx<nXc;statesIdx++) {
      x[statesIdx] = x0[statesIdx] + deltaY[statesIdx];
    }

    rtsiSetSimTimeStep(si,MAJOR_TIME_STEP);
  }

  /* Model step function */
  void UGV_Arduino_code_step(void)
  {
    /* local block i/o variables */
    real_T rtb_Subtract;
    real_T rtb_Switch1;
    int_T iy;
    uint16_T rtb_SpeedsensorInput_0;
    int16_T jmax;
    int16_T jj;
    int16_T b_j;
    int16_T ix;
    int16_T k;
    uint16_T rtb_Switch2;
    real_T rtb_DigitalClock;
    real_T f_idx_0;
    real_T f_idx_1;
    uint8_T rtb_UnitConversion_0;
    boolean_T exitg1;
    if (rtmIsMajorTimeStep(UGV_Arduino_code_M)) {
      /* set solver stop time */
      if (!(UGV_Arduino_code_M->Timing.clockTick0+1)) {
        rtsiSetSolverStopTime(&UGV_Arduino_code_M->solverInfo,
                              ((UGV_Arduino_code_M->Timing.clockTickH0 + 1) *
          UGV_Arduino_code_M->Timing.stepSize0 * 4294967296.0));
      } else {
        rtsiSetSolverStopTime(&UGV_Arduino_code_M->solverInfo,
                              ((UGV_Arduino_code_M->Timing.clockTick0 + 1) *
          UGV_Arduino_code_M->Timing.stepSize0 +
          UGV_Arduino_code_M->Timing.clockTickH0 *
          UGV_Arduino_code_M->Timing.stepSize0 * 4294967296.0));
      }
    }                                    /* end MajorTimeStep */

    /* Update absolute time of base rate at minor time step */
    if (rtmIsMinorTimeStep(UGV_Arduino_code_M)) {
      UGV_Arduino_code_M->Timing.t[0] = rtsiGetT(&UGV_Arduino_code_M->solverInfo);
    }

    /* StateSpace: '<Root>/Analog Filter Design' */
    UGV_Arduino_code_B.DataTypeConversion2 =
      UGV_Arduino_code_P.AnalogFilterDesign_C *
      UGV_Arduino_code_X.AnalogFilterDesign_CSTATE[2];
    if (rtmIsMajorTimeStep(UGV_Arduino_code_M)) {
      /* S-Function (arduinoanaloginput_sfcn): '<Root>/Speed Command' */
      UGV_Arduino_code_B.Speed_Setpoint = MW_analogRead
        (UGV_Arduino_code_P.SpeedCommand_p1);

      /* Switch: '<S8>/Switch' incorporates:
       *  Constant: '<Root>/Constant9'
       *  RelationalOperator: '<S8>/UpperRelop'
       */
      if (UGV_Arduino_code_B.Speed_Setpoint < UGV_Arduino_code_P.Constant9_Value)
      {
        rtb_DigitalClock = floor(UGV_Arduino_code_P.Constant9_Value);
        if (rtIsNaN(rtb_DigitalClock) || rtIsInf(rtb_DigitalClock)) {
          rtb_DigitalClock = 0.0;
        } else {
          rtb_DigitalClock = fmod(rtb_DigitalClock, 65536.0);
        }

        UGV_Arduino_code_B.Switch = rtb_DigitalClock < 0.0 ? (uint16_T)-(int16_T)
          (uint16_T)-rtb_DigitalClock : (uint16_T)rtb_DigitalClock;
      } else {
        UGV_Arduino_code_B.Switch = UGV_Arduino_code_B.Speed_Setpoint;
      }

      /* End of Switch: '<S8>/Switch' */
    }

    /* Switch: '<S8>/Switch2' incorporates:
     *  RelationalOperator: '<S8>/LowerRelop1'
     */
    if (UGV_Arduino_code_B.Speed_Setpoint > UGV_Arduino_code_B.DataTypeConversion2)
    {
      rtb_DigitalClock = floor(UGV_Arduino_code_B.DataTypeConversion2);
      if (rtIsInf(rtb_DigitalClock)) {
        rtb_DigitalClock = 0.0;
      } else {
        rtb_DigitalClock = fmod(rtb_DigitalClock, 65536.0);
      }

      rtb_Switch2 = rtb_DigitalClock < 0.0 ? (uint16_T)-(int16_T)(uint16_T)
        -rtb_DigitalClock : (uint16_T)rtb_DigitalClock;
      rtb_DigitalClock = floor(UGV_Arduino_code_B.DataTypeConversion2);
      if (rtIsInf(rtb_DigitalClock)) {
        rtb_DigitalClock = 0.0;
      } else {
        rtb_DigitalClock = fmod(rtb_DigitalClock, 65536.0);
      }

      /* DataTypeConversion: '<Root>/Data Type Conversion2' */
      UGV_Arduino_code_B.DataTypeConversion2 = rtb_DigitalClock < 0.0 ? (uint16_T)
        -(int16_T)(uint16_T)-rtb_DigitalClock : (uint16_T)rtb_DigitalClock;
    } else {
      rtb_Switch2 = UGV_Arduino_code_B.Switch;

      /* DataTypeConversion: '<Root>/Data Type Conversion2' */
      UGV_Arduino_code_B.DataTypeConversion2 = UGV_Arduino_code_B.Switch;
    }

    /* End of Switch: '<S8>/Switch2' */
    if (rtmIsMajorTimeStep(UGV_Arduino_code_M)) {
      /* S-Function (arduinoanaloginput_sfcn): '<Root>/Speed sensor Input' */
      rtb_SpeedsensorInput_0 = MW_analogRead
        (UGV_Arduino_code_P.SpeedsensorInput_p1);

      /* Outputs for Triggered SubSystem: '<S3>/Encorder pulses to rev//min to m//s' 
  incorporates:
       *  TriggerPort: '<S11>/Trigger'
       */
      if (rtmIsMajorTimeStep(UGV_Arduino_code_M)) {
        /* S-Function (arduinoanaloginput_sfcn): '<Root>/Speed sensor Input' */
        if ((rtb_SpeedsensorInput_0 > 0U) &&
            (UGV_Arduino_code_PrevZCX.Encorderpulsestorevmintoms_Trig != POS_ZCSIG))
        {
          /* DigitalClock: '<S11>/Digital Clock' */
          rtb_DigitalClock = (((UGV_Arduino_code_M->Timing.clockTick1+
                              UGV_Arduino_code_M->Timing.clockTickH1*
```

```
                            4294967296.0)) * 0.02);

      /* Sum: '<S11>/Subtract' incorporates:
       *  UnitDelay: '<S11>/Unit Delay'
       */
      rtb_Subtract = rtb_DigitalClock - UGV_Arduino_code_DW.UnitDelay_DSTATE;

      /* Gain: '<S11>/Gain' incorporates:
       *  Constant: '<S11>/Constant1'
       *  Constant: '<S11>/Radius(m)'
       *  Product: '<S11>/Divide'
       *  Product: '<S11>/Product'
       */
      UGV_Arduino_code_B.Gain = UGV_Arduino_code_P.Constant1_Value /
        rtb_Subtract * UGV_Arduino_code_P.Radiusm_Value *
        UGV_Arduino_code_P.Gain_Gain;

      /* Update for UnitDelay: '<S11>/Unit Delay' */
      UGV_Arduino_code_DW.UnitDelay_DSTATE = rtb_DigitalClock;
    }

    UGV_Arduino_code_PrevZCX.Encorderpulsestorevmintoms_Trig = (ZCSigState)
      (rtb_SpeedsensorInput_0 > 0U);
  }

  /* End of Outputs for SubSystem: '<S3>/Encorder pulses to rev//min to m//s' */
}

/* Gain: '<S5>/Filter Coefficient' incorporates:
 *  DataTypeConversion: '<Root>/Data Type Conversion2'
 *  Gain: '<S5>/Derivative Gain'
 *  Gain: '<S5>/Setpoint Weighting (Derivative)'
 *  Integrator: '<S5>/Filter'
 *  Sum: '<S5>/Sum3'
 *  Sum: '<S5>/SumD'
 */
UGV_Arduino_code_B.FilterCoefficient =
  ((UGV_Arduino_code_P.PIDController2DOF1_c * (real_T)rtb_Switch2 -
    UGV_Arduino_code_B.Gain) * UGV_Arduino_code_P.PIDController2DOF1_D -
   UGV_Arduino_code_X.Filter_CSTATE) * UGV_Arduino_code_P.PIDController2DOF1_N;

/* DataTypeConversion: '<S6>/Data Type Conversion' incorporates:
 *  DataTypeConversion: '<Root>/Data Type Conversion2'
 *  Gain: '<S5>/Proportional Gain'
 *  Gain: '<S5>/Setpoint Weighting (Proportional)'
 *  Integrator: '<S5>/Integrator'
 *  Sum: '<S5>/Sum'
 *  Sum: '<S5>/Sum1'
 */
rtb_DigitalClock = ((UGV_Arduino_code_P.PIDController2DOF1_b * (real_T)
                     rtb_Switch2 - UGV_Arduino_code_B.Gain) *
                    UGV_Arduino_code_P.PIDController2DOF1_P +
                    UGV_Arduino_code_X.Integrator_CSTATE) +
  UGV_Arduino_code_B.FilterCoefficient;
if (rtb_DigitalClock < 256.0) {
  if (rtb_DigitalClock >= 0.0) {
    UGV_Arduino_code_B.DataTypeConversion = (uint8_T)rtb_DigitalClock;
  } else {
    UGV_Arduino_code_B.DataTypeConversion = 0U;
  }
} else {
  UGV_Arduino_code_B.DataTypeConversion = MAX_uint8_T;
}
```

```
  /* End of DataTypeConversion: '<S6>/Data Type Conversion' */
  if (rtmIsMajorTimeStep(UGV_Arduino_code_M)) {
    /* S-Function (arduinoanalogoutput_sfcn): '<S6>/PWM' */
    MW_analogWrite(UGV_Arduino_code_P.PWM_pinNumber,
                   UGV_Arduino_code_B.DataTypeConversion);

    /* S-Function (arduinoanaloginput_sfcn): '<Root>/From ADXL337  Accelaration Sensor
' */
    rtb_Switch2 = MW_analogRead
      (UGV_Arduino_code_P.FromADXL337AccelarationSensor_p);

    /* Delay: '<S7>/delayTheta' incorporates:
     *  Constant: '<S7>/InitialParameters'
     */
    if (UGV_Arduino_code_DW.icLoad != 0) {
      UGV_Arduino_code_DW.estimatedParameters[0] =
        UGV_Arduino_code_P.InitialParameters_Value[0];
      UGV_Arduino_code_DW.estimatedParameters[1] =
        UGV_Arduino_code_P.InitialParameters_Value[1];
    }

    /* Delay: '<S7>/delayL' incorporates:
     *  Constant: '<S7>/InitialCovariance'
     */
    if (UGV_Arduino_code_DW.icLoad_b != 0) {
      UGV_Arduino_code_DW.P[0] = UGV_Arduino_code_P.InitialCovariance_Value[0];
      UGV_Arduino_code_DW.P[1] = UGV_Arduino_code_P.InitialCovariance_Value[1];
      UGV_Arduino_code_DW.P[2] = UGV_Arduino_code_P.InitialCovariance_Value[2];
      UGV_Arduino_code_DW.P[3] = UGV_Arduino_code_P.InitialCovariance_Value[3];
    }

    /* SignalConversion: '<S26>/TmpSignal ConversionAt SFunction Inport1'
incorporates:
     *  Constant: '<Root>/Constant1'
     *  Constant: '<Root>/Constant3'
     *  Constant: '<Root>/UGV_Height '
     *  Gain: '<Root>/Gain'
     *  Gain: '<Root>/g'
     *  MATLAB Function: '<S7>/RLS'
     *  Product: '<Root>/Divide3'
     *  Sum: '<Root>/Add'
     */
    UGV_Arduino_code_B.TmpSignalConversionAtSFunct[0] =
      UGV_Arduino_code_P.g_Gain * UGV_Arduino_code_P.Constant3_Value /
      ((UGV_Arduino_code_P.Constant1_Value_f +
        UGV_Arduino_code_P.UGV_Height_Value) * UGV_Arduino_code_P.Gain_Gain_i);

    /* MATLAB Function: '<S7>/RLS' incorporates:
     *  Bias: '<S10>/Bias1'
     *  Constant: '<Root>/Constant2'
     *  Constant: '<S10>/Constant1'
     *  Constant: '<S7>/AdaptationParameter1'
     *  Constant: '<S7>/Enable'
     *  DataTypeConversion: '<S7>/DataTypeConversionEnable'
     *  Delay: '<S7>/delayL'
     *  Delay: '<S7>/delayTheta'
     *  Product: '<S10>/Divide'
     *  S-Function (arduinoanaloginput_sfcn): '<Root>/From ADXL337  Accelaration
Sensor '
     */
    /* MATLAB Function 'Estimators/Recursive Least Squares Estimator/RLS': '<S26>:1'
*/
    /*    Copyright 2013-2013 The MathWorks, Inc. */
    /* '<S26>:1:7' coder.extrinsic('idrlsstep_double_mex'); */
    /* '<S26>:1:8' coder.extrinsic('idrlsstep_single_mex'); */
    /* '<S26>:1:9' if coder.target('Sfun') && ~coder.target('RtwForRapid') &&
~coder.target('RtwForSim') */
    /* '<S26>:1:22' else */
    /*  Use M code for code generation */
    /* '<S26>:1:24' [thetaNew,estimationError,PNew] = ... */
    /* '<S26>:1:25'
idrlsstep(uMeas,yMeas,isEnabled,adg1,adg2,theta,P,algorithmParams); */
    UGV_Arduino_code_B.PNew[0] = UGV_Arduino_code_DW.P[0];
    UGV_Arduino_code_B.PNew[1] = UGV_Arduino_code_DW.P[1];
    UGV_Arduino_code_B.PNew[2] = UGV_Arduino_code_DW.P[2];
    UGV_Arduino_code_B.PNew[3] = UGV_Arduino_code_DW.P[3];
    UGV_Arduino_code_B.thetaNew[0] = UGV_Arduino_code_DW.estimatedParameters[0];
    UGV_Arduino_code_B.thetaNew[1] = UGV_Arduino_code_DW.estimatedParameters[1];
    UGV_Arduino_code_B.UnitConversion = (real_T)(rtb_Switch2 +
      UGV_Arduino_code_P.Bias1_Bias) / UGV_Arduino_code_P.Constant1_Value_b -
      (UGV_Arduino_code_B.TmpSignalConversionAtSFunct[0] *
       UGV_Arduino_code_DW.estimatedParameters[0] +
       UGV_Arduino_code_P.Constant2_Value *
       UGV_Arduino_code_DW.estimatedParameters[1]);
    if (UGV_Arduino_code_P.Enable_Value != 0.0) {
      UGV_Arduino_code_B.L[0] = UGV_Arduino_code_DW.P[0];
      UGV_Arduino_code_B.L[1] = UGV_Arduino_code_DW.P[1];
      UGV_Arduino_code_B.L[2] = UGV_Arduino_code_DW.P[2];
      UGV_Arduino_code_B.L[3] = UGV_Arduino_code_DW.P[3];
      f_idx_0 = UGV_Arduino_code_B.TmpSignalConversionAtSFunct[0] *
        UGV_Arduino_code_DW.P[0] + UGV_Arduino_code_P.Constant2_Value *
        UGV_Arduino_code_DW.P[1];
      f_idx_1 = UGV_Arduino_code_B.TmpSignalConversionAtSFunct[0] *
        UGV_Arduino_code_DW.P[2] + UGV_Arduino_code_P.Constant2_Value *
        UGV_Arduino_code_DW.P[3];
      UGV_Arduino_code_B.TmpSignalConversionAtSFunct[0] = 0.0;
      UGV_Arduino_code_B.thetaNew[0] = 0.0;
      UGV_Arduino_code_B.TmpSignalConversionAtSFunct[1] = 0.0;
      UGV_Arduino_code_B.thetaNew[1] = 0.0;
      rtb_DigitalClock = f_idx_1 * f_idx_1 + 1.0;
      UGV_Arduino_code_B.Rollover_Index_Estimation = sqrt(1.0 / rtb_DigitalClock);
      UGV_Arduino_code_B.sqrtA0timesA1 =
        UGV_Arduino_code_B.Rollover_Index_Estimation * rtb_DigitalClock;
      jj = 0;
      while (jj <= 0) {
        UGV_Arduino_code_B.thetaNew[1] = UGV_Arduino_code_B.L[3] * f_idx_1 +
          UGV_Arduino_code_B.TmpSignalConversionAtSFunct[1];
        UGV_Arduino_code_B.L[3] = UGV_Arduino_code_B.L[3] *
          UGV_Arduino_code_B.Rollover_Index_Estimation -
          UGV_Arduino_code_B.TmpSignalConversionAtSFunct[1] * f_idx_1 /
          UGV_Arduino_code_B.sqrtA0timesA1;
        UGV_Arduino_code_B.TmpSignalConversionAtSFunct[1] =
          UGV_Arduino_code_B.thetaNew[1];
        jj = 1;
      }

      UGV_Arduino_code_B.Rollover_Index_Estimation = rtb_DigitalClock;
      rtb_DigitalClock += f_idx_0 * f_idx_0;
      UGV_Arduino_code_B.Rollover_Index_Estimation = sqrt
        (UGV_Arduino_code_B.Rollover_Index_Estimation / rtb_DigitalClock);
      UGV_Arduino_code_B.sqrtA0timesA1 =
        UGV_Arduino_code_B.Rollover_Index_Estimation * rtb_DigitalClock;
      for (jj = 0; jj < 2; jj++) {
        UGV_Arduino_code_B.thetaNew[1 - jj] = UGV_Arduino_code_B.L[1 - jj] *
          f_idx_0 + UGV_Arduino_code_B.TmpSignalConversionAtSFunct[1 - jj];
        UGV_Arduino_code_B.L[1 - jj] = UGV_Arduino_code_B.L[1 - jj] *
          UGV_Arduino_code_B.Rollover_Index_Estimation -
          UGV_Arduino_code_B.TmpSignalConversionAtSFunct[1 - jj] * f_idx_0 /
          UGV_Arduino_code_B.sqrtA0timesA1;
        UGV_Arduino_code_B.TmpSignalConversionAtSFunct[1 - jj] =
          UGV_Arduino_code_B.thetaNew[1 - jj];
      }

      for (jj = 0; jj < 2; jj++) {
        UGV_Arduino_code_B.thetaNew[jj] /= rtb_DigitalClock;
        UGV_Arduino_code_B.PNew[jj] = (UGV_Arduino_code_B.L[jj + 2] *
          UGV_Arduino_code_B.L[2] + UGV_Arduino_code_B.L[jj] *
          UGV_Arduino_code_B.L[0]) +
          UGV_Arduino_code_P.AdaptationParameter1_Value[jj];
        UGV_Arduino_code_B.PNew[jj + 2] = (UGV_Arduino_code_B.L[jj + 2] *
          UGV_Arduino_code_B.L[3] + UGV_Arduino_code_B.L[jj] *
          UGV_Arduino_code_B.L[1]) +
          UGV_Arduino_code_P.AdaptationParameter1_Value[jj + 2];
      }

      jmax = 0;
      b_j = 0;
      exitg1 = false;
      while ((!exitg1) && (b_j + 1 < 3)) {
        jj = (b_j << 1) + b_j;
        rtb_DigitalClock = 0.0;
        if (!(b_j < 1)) {
          ix = b_j;
          iy = b_j;
          k = 1;
          while (k <= b_j) {
            rtb_DigitalClock += UGV_Arduino_code_B.PNew[ix] *
              UGV_Arduino_code_B.PNew[iy];
            ix += 2;
            iy += 2;
            k = 2;
          }
        }

        rtb_DigitalClock = UGV_Arduino_code_B.PNew[jj] - rtb_DigitalClock;
        if (rtb_DigitalClock > 0.0) {
          rtb_DigitalClock = sqrt(rtb_DigitalClock);
          UGV_Arduino_code_B.PNew[jj] = rtb_DigitalClock;
          if (b_j + 1 < 2) {
            rtb_DigitalClock = 1.0 / rtb_DigitalClock;
            for (iy = jj + 1; iy + 1 <= jj + 2; iy++) {
              UGV_Arduino_code_B.PNew[iy] *= rtb_DigitalClock;
            }
          }

          b_j++;
        } else {
          UGV_Arduino_code_B.PNew[jj] = rtb_DigitalClock;
          jmax = b_j + 1;
          exitg1 = true;
        }
      }

      if (jmax == 0) {
        jmax = 3;
      }

      jmax--;
      b_j = 2;
      while (b_j <= jmax) {
        UGV_Arduino_code_B.PNew[2] = 0.0;
        b_j = 3;
      }

      UGV_Arduino_code_B.thetaNew[0] = UGV_Arduino_code_B.thetaNew[0] *
        UGV_Arduino_code_B.UnitConversion +
        UGV_Arduino_code_DW.estimatedParameters[0];
      UGV_Arduino_code_B.thetaNew[1] = UGV_Arduino_code_B.thetaNew[1] *
        UGV_Arduino_code_B.UnitConversion +
        UGV_Arduino_code_DW.estimatedParameters[1];
    }

    /* Trigonometry: '<Root>/Trigonometric Function1' */
    rtb_DigitalClock = atan(UGV_Arduino_code_B.thetaNew[1]);

    /* UnitConversion: '<S1>/Unit Conversion' */
    /* Unit Conversion - from: rad to: deg
       Expression: output = (57.2958*input) + (0) */
    UGV_Arduino_code_B.UnitConversion = 57.295779513082323 * rtb_DigitalClock;

    /* Switch: '<Root>/Desired Roll Angle' incorporates:
     *  Abs: '<Root>/Abs2'
     *  Constant: '<Root>/Constant7'
     *  Constant: '<Root>/Constant8'
     *  Gain: '<Root>/Gain2'
     *  Sum: '<Root>/Add2'
     */
    if (fabs(UGV_Arduino_code_B.UnitConversion) >=
        UGV_Arduino_code_P.DesiredRollAngle_Threshold) {
      UGV_Arduino_code_B.UnitConversion = UGV_Arduino_code_P.Gain2_Gain *
        UGV_Arduino_code_B.UnitConversion + UGV_Arduino_code_P.Constant7_Value;
    } else {
      UGV_Arduino_code_B.UnitConversion = UGV_Arduino_code_P.Constant8_Value;
    }

    /* End of Switch: '<Root>/Desired Roll Angle' */

    /* Gain: '<S4>/Gain1' */
    UGV_Arduino_code_B.UnitConversion *= UGV_Arduino_code_P.Gain1_Gain_l;

    /* DataTypeConversion: '<S9>/Data Type Conversion' */
    if (UGV_Arduino_code_B.UnitConversion < 256.0) {
      if (UGV_Arduino_code_B.UnitConversion >= 0.0) {
        rtb_UnitConversion_0 = (uint8_T)UGV_Arduino_code_B.UnitConversion;
      } else {
        rtb_UnitConversion_0 = 0U;
      }
    } else {
      rtb_UnitConversion_0 = MAX_uint8_T;
    }

    /* End of DataTypeConversion: '<S9>/Data Type Conversion' */

    /* S-Function (arduinoservowrite_sfcn): '<S9>/Servo Write' */
    MW_servoWrite(UGV_Arduino_code_P.ServoWrite_p1, rtb_UnitConversion_0);

    /* Product: '<Root>/Product' incorporates:
     *  Trigonometry: '<Root>/Trigonometric Function2'
     */
    UGV_Arduino_code_B.Rollover_Index_Estimation = UGV_Arduino_code_B.thetaNew[0]
      * cos(rtb_DigitalClock);

    /* Abs: '<Root>/Abs1' */
    UGV_Arduino_code_B.UnitConversion = fabs
      (UGV_Arduino_code_B.Rollover_Index_Estimation);
    /* Memory: '<Root>/Memory' */
    UGV_Arduino_code_B.Memory = UGV_Arduino_code_DW.Memory_PreviousInput;

    /* Switch: '<Root>/Switch1' incorporates:
     *  Abs: '<Root>/Abs'
     *  Constant: '<Root>/Constant5'
     *  Constant: '<Root>/Constant6'
     *  Gain: '<Root>/Gain1'
     *  Sum: '<Root>/Add1'
     */
    if (UGV_Arduino_code_B.UnitConversion >=
        UGV_Arduino_code_P.Switch1_Threshold) {
      rtb_Switch1 = UGV_Arduino_code_P.Constant5_Value - fabs
        (UGV_Arduino_code_P.Gain1_Gain *
         UGV_Arduino_code_B.Rollover_Index_Estimation);
    } else {
      rtb_Switch1 = UGV_Arduino_code_P.Constant6_Value;
    }

    /* End of Switch: '<Root>/Switch1' */
  }

  /* Gain: '<S5>/Integral Gain' incorporates:
   *  Sum: '<S5>/Sum2'
   */
  UGV_Arduino_code_B.IntegralGain = (UGV_Arduino_code_B.DataTypeConversion2 -
    UGV_Arduino_code_B.Gain) * UGV_Arduino_code_P.PIDController2DOF1_I;
  if (rtmIsMajorTimeStep(UGV_Arduino_code_M)) {
    if (rtmIsMajorTimeStep(UGV_Arduino_code_M)) {
      /* Update for Delay: '<S7>/delayTheta' */
      UGV_Arduino_code_DW.icLoad = 0U;
      UGV_Arduino_code_DW.estimatedParameters[0] = UGV_Arduino_code_B.thetaNew[0];
      UGV_Arduino_code_DW.estimatedParameters[1] = UGV_Arduino_code_B.thetaNew[1];

      /* Update for Delay: '<S7>/delayL' */
      UGV_Arduino_code_DW.icLoad_b = 0U;
      UGV_Arduino_code_DW.P[0] = UGV_Arduino_code_B.PNew[0];
      UGV_Arduino_code_DW.P[1] = UGV_Arduino_code_B.PNew[1];
      UGV_Arduino_code_DW.P[2] = UGV_Arduino_code_B.PNew[2];
      UGV_Arduino_code_DW.P[3] = UGV_Arduino_code_B.PNew[3];

      /* Update for Memory: '<Root>/Memory' */
      UGV_Arduino_code_DW.Memory_PreviousInput = rtb_Switch1;
    }
  }                                    /* end MajorTimeStep */

  if (rtmIsMajorTimeStep(UGV_Arduino_code_M)) {
    rt_ertODEUpdateContinuousStates(&UGV_Arduino_code_M->solverInfo);

    /* Update absolute time for base rate */
    /* The "clockTick0" counts the number of times the code of this task has
     * been executed. The absolute time is the multiplication of "clockTick0"
     * and "Timing.stepSize0". Size of "clockTick0" ensures timer will not
     * overflow during the application lifespan selected.
     * Timer of this task consists of two 32 bit unsigned integers.
     * The two integers represent the low bits Timing.clockTick0 and the high bits
     * Timing.clockTickH0. When the low bit overflows to 0, the high bits increment.
     */
    if (!(++UGV_Arduino_code_M->Timing.clockTick0)) {
      ++UGV_Arduino_code_M->Timing.clockTickH0;
    }

    UGV_Arduino_code_M->Timing.t[0] = rtsiGetSolverStopTime
      (&UGV_Arduino_code_M->solverInfo);
    {
      /* Update absolute timer for sample time: [0.02s, 0.0s] */
      /* The "clockTick1" counts the number of times the code of this task has
       * been executed. The resolution of this integer timer is 0.02, which is the
step size
       * of the task. Size of "clockTick1" ensures timer will not overflow during the
       * application lifespan selected.
       * Timer of this task consists of two 32 bit unsigned integers.
       * The two integers represent the low bits Timing.clockTick1 and the high bits
       * Timing.clockTickH1. When the low bit overflows to 0, the high bits increment.
       */
      UGV_Arduino_code_M->Timing.clockTick1++;
      if (!UGV_Arduino_code_M->Timing.clockTick1) {
        UGV_Arduino_code_M->Timing.clockTickH1++;
      }
    }
  }                                    /* end MajorTimeStep */
}

/* Derivatives for root system: '<Root>' */
void UGV_Arduino_code_derivatives(void)
{
  XDot_UGV_Arduino_code_T *_rtXdot;
  _rtXdot = ((XDot_UGV_Arduino_code_T *) UGV_Arduino_code_M->derivs);

  /* Derivatives for StateSpace: '<Root>/Analog Filter Design' */
  _rtXdot->AnalogFilterDesign_CSTATE[0] = 0.0;
  _rtXdot->AnalogFilterDesign_CSTATE[1] = 0.0;
  _rtXdot->AnalogFilterDesign_CSTATE[2] = 0.0;
  _rtXdot->AnalogFilterDesign_CSTATE[0] +=
    UGV_Arduino_code_P.AnalogFilterDesign_A[0] *
    UGV_Arduino_code_X.AnalogFilterDesign_CSTATE[0];
  _rtXdot->AnalogFilterDesign_CSTATE[1] +=
    UGV_Arduino_code_P.AnalogFilterDesign_A[1] *
    UGV_Arduino_code_X.AnalogFilterDesign_CSTATE[0];
  _rtXdot->AnalogFilterDesign_CSTATE[1] +=
    UGV_Arduino_code_P.AnalogFilterDesign_A[2] *
    UGV_Arduino_code_X.AnalogFilterDesign_CSTATE[1];
  _rtXdot->AnalogFilterDesign_CSTATE[1] +=
    UGV_Arduino_code_P.AnalogFilterDesign_A[3] *
    UGV_Arduino_code_X.AnalogFilterDesign_CSTATE[2];
  _rtXdot->AnalogFilterDesign_CSTATE[2] +=
    UGV_Arduino_code_P.AnalogFilterDesign_A[4] *
    UGV_Arduino_code_X.AnalogFilterDesign_CSTATE[1];
  _rtXdot->AnalogFilterDesign_CSTATE[0] +=
    UGV_Arduino_code_P.AnalogFilterDesign_B * UGV_Arduino_code_B.Memory;

  /* Derivatives for Integrator: '<S5>/Integrator' */
  _rtXdot->Integrator_CSTATE = UGV_Arduino_code_B.IntegralGain;

  /* Derivatives for Integrator: '<S5>/Filter' */
  _rtXdot->Filter_CSTATE = UGV_Arduino_code_B.FilterCoefficient;
}

/* Model initialize function */
void UGV_Arduino_code_initialize(void)
{
  /* Registration code */

  /* initialize non-finites */
  rt_InitInfAndNaN(sizeof(real_T));

  /* initialize real-time model */
```

```
(void) memset((void *)UGV_Arduino_code_M, 0,
              sizeof(RT_MODEL_UGV_Arduino_code_T));

{
  /* Setup solver object */
  rtsiSetSimTimeStepPtr(&UGV_Arduino_code_M->solverInfo,
                        &UGV_Arduino_code_M->Timing.simTimeStep);
  rtsiSetTPtr(&UGV_Arduino_code_M->solverInfo, &rtmGetTPtr(UGV_Arduino_code_M));
  rtsiSetStepSizePtr(&UGV_Arduino_code_M->solverInfo,
                     &UGV_Arduino_code_M->Timing.stepSize0);
  rtsiSetdXPtr(&UGV_Arduino_code_M->solverInfo, &UGV_Arduino_code_M->derivs);
  rtsiSetContStatesPtr(&UGV_Arduino_code_M->solverInfo, (real_T **)
                       &UGV_Arduino_code_M->contStates);
  rtsiSetNumContStatesPtr(&UGV_Arduino_code_M->solverInfo,
    &UGV_Arduino_code_M->Sizes.numContStates);
  rtsiSetNumPeriodicContStatesPtr(&UGV_Arduino_code_M->solverInfo,
    &UGV_Arduino_code_M->Sizes.numPeriodicContStates);
  rtsiSetPeriodicContStateIndicesPtr(&UGV_Arduino_code_M->solverInfo,
    &UGV_Arduino_code_M->periodicContStateIndices);
  rtsiSetPeriodicContStateRangesPtr(&UGV_Arduino_code_M->solverInfo,
    &UGV_Arduino_code_M->periodicContStateRanges);
  rtsiSetErrorStatusPtr(&UGV_Arduino_code_M->solverInfo, (&rtmGetErrorStatus
    (UGV_Arduino_code_M)));
  rtsiSetRTModelPtr(&UGV_Arduino_code_M->solverInfo, UGV_Arduino_code_M);
}

rtsiSetSimTimeStep(&UGV_Arduino_code_M->solverInfo, MAJOR_TIME_STEP);
UGV_Arduino_code_M->intgData.deltaY= UGV_Arduino_code_M->OdeDeltaY;
UGV_Arduino_code_M->intgData.f[0] = UGV_Arduino_code_M->odeF[0];
UGV_Arduino_code_M->intgData.f[1] = UGV_Arduino_code_M->odeF[1];
UGV_Arduino_code_M->intgData.f[2] = UGV_Arduino_code_M->odeF[2];
UGV_Arduino_code_M->intgData.f[3] = UGV_Arduino_code_M->odeF[3];
UGV_Arduino_code_M->intgData.f[4] = UGV_Arduino_code_M->odeF[4];
UGV_Arduino_code_M->intgData.f[5] = UGV_Arduino_code_M->odeF[5];
UGV_Arduino_code_M->intgData.f[6] = UGV_Arduino_code_M->odeF[6];
UGV_Arduino_code_M->intgData.f[7] = UGV_Arduino_code_M->odeF[7];
UGV_Arduino_code_M->intgData.f[8] = UGV_Arduino_code_M->odeF[8];
UGV_Arduino_code_M->intgData.f[9] = UGV_Arduino_code_M->odeF[9];
UGV_Arduino_code_M->intgData.f[10] = UGV_Arduino_code_M->odeF[10];
UGV_Arduino_code_M->intgData.f[11] = UGV_Arduino_code_M->odeF[11];
UGV_Arduino_code_M->intgData.f[12] = UGV_Arduino_code_M->odeF[12];
UGV_Arduino_code_M->intgData.x0 = UGV_Arduino_code_M->odeX0;
UGV_Arduino_code_M->contStates = ((X_UGV_Arduino_code_T *) &UGV_Arduino_code_X);
rtsiSetSolverData(&UGV_Arduino_code_M->solverInfo, (void *)
                  &UGV_Arduino_code_M->intgData);
rtsiSetSolverName(&UGV_Arduino_code_M->solverInfo,"ode8");
rtmSetTPtr(UGV_Arduino_code_M, &UGV_Arduino_code_M->Timing.tArray[0]);
UGV_Arduino_code_M->Timing.stepSize0 = 0.02;

/* block I/O */
(void) memset(((void *) &UGV_Arduino_code_B), 0,
              sizeof(B_UGV_Arduino_code_T));

/* states (continuous) */
{
  (void) memset((void *)&UGV_Arduino_code_X, 0,
                sizeof(X_UGV_Arduino_code_T));
}

/* states (dwork) */
(void) memset((void *)&UGV_Arduino_code_DW, 0,
              sizeof(DW_UGV_Arduino_code_T));

/* Start for S-Function (arduinoanaloginput_sfcn): '<Root>/Speed Command' */
```

```
  MW_pinModeAnalogInput(UGV_Arduino_code_P.SpeedCommand_p1);

  /* Start for S-Function (arduinoanaloginput_sfcn): '<Root>/Speed sensor Input' */
  MW_pinModeAnalogInput(UGV_Arduino_code_P.SpeedsensorInput_p1);

  /* Start for S-Function (arduinoanalogoutput_sfcn): '<S6>/PWM' */
  MW_pinModeOutput(UGV_Arduino_code_P.PWM_pinNumber);

  /* Start for S-Function (arduinoanaloginput_sfcn): '<Root>/From ADXL337
Accelaration Sensor ' */
  MW_pinModeAnalogInput(UGV_Arduino_code_P.FromADXL337AccelarationSensor_p);

  /* Start for S-Function (arduinoservowrite_sfcn): '<S9>/Servo Write' */
  MW_servoAttach(UGV_Arduino_code_P.ServoWrite_p1,
                 UGV_Arduino_code_P.ServoWrite_pinNumber);
  UGV_Arduino_code_PrevZCX.Encorderpulsestorevmintoms_Trig = POS_ZCSIG;

  /* InitializeConditions for StateSpace: '<Root>/Analog Filter Design' */
  UGV_Arduino_code_X.AnalogFilterDesign_CSTATE[0] =
    UGV_Arduino_code_P.AnalogFilterDesign_X0;
  UGV_Arduino_code_X.AnalogFilterDesign_CSTATE[1] =
    UGV_Arduino_code_P.AnalogFilterDesign_X0;
  UGV_Arduino_code_X.AnalogFilterDesign_CSTATE[2] =
    UGV_Arduino_code_P.AnalogFilterDesign_X0;

  /* InitializeConditions for Integrator: '<S5>/Integrator' */
  UGV_Arduino_code_X.Integrator_CSTATE = UGV_Arduino_code_P.Integrator_IC;

  /* InitializeConditions for Integrator: '<S5>/Filter' */
  UGV_Arduino_code_X.Filter_CSTATE = UGV_Arduino_code_P.Filter_IC;

  /* InitializeConditions for Delay: '<S7>/delayTheta' */
  UGV_Arduino_code_DW.icLoad = 1U;

  /* InitializeConditions for Delay: '<S7>/delayL' */
  UGV_Arduino_code_DW.icLoad_b = 1U;

  /* InitializeConditions for Memory: '<Root>/Memory' */
  UGV_Arduino_code_DW.Memory_PreviousInput = UGV_Arduino_code_P.Memory_X0;

  /* SystemInitialize for Triggered SubSystem: '<S3>/Encorder pulses to rev//min to
m//s' */
  /* InitializeConditions for UnitDelay: '<S11>/Unit Delay' */
  UGV_Arduino_code_DW.UnitDelay_DSTATE =
    UGV_Arduino_code_P.UnitDelay_InitialCondition;

  /* SystemInitialize for Outport: '<S11>/Out1' */
  UGV_Arduino_code_B.Gain = UGV_Arduino_code_P.Out1_Y0;

  /* End of SystemInitialize for SubSystem: '<S3>/Encorder pulses to rev//min to m//s'
*/
}

/* Model terminate function */
void UGV_Arduino_code_terminate(void)
{
  /* (no terminate code required) */
}
```

Appendix B.1: Header files for Arduino board

```
/*
 * UGV_Arduino_code.h
 *
 * Student License - for use by students to meet course requirements and
 * perform academic research at degree granting institutions only.  Not
 * for government, commercial, or other organizational use.
 *
 * Code generation for model "UGV_Arduino_code".
 *
 * Model version              : 1.10
 * Simulink Coder version : 8.11 (R2016b) 25-Aug-2016
 * C source code generated on : Mon Mar 19 22:15:44 2018
 *
 * Target selection: ert.tlc
 * Note: GRT includes extra infrastructure and instrumentation for prototyping
 * Embedded hardware selection: Atmel->AVR
 * Code generation objectives: Unspecified
 * Validation result: Not run
 */

#ifndef RTW_HEADER_UGV_Arduino_code_h_
#define RTW_HEADER_UGV_Arduino_code_h_
#include <math.h>
#include <string.h>
#include <stddef.h>
#ifndef UGV_Arduino_code_COMMON_INCLUDES_
# define UGV_Arduino_code_COMMON_INCLUDES_
#include "rtwtypes.h"
#include "rtw_continuous.h"
#include "rtw_solver.h"
#include "arduino_analoginput_lct.h"
#include "arduino_analogoutput_lct.h"
#include "arduino_servowrite_lct.h"
#endif                                 /* UGV_Arduino_code_COMMON_INCLUDES_ */

#include "UGV_Arduino_code_types.h"

/* Shared type includes */
#include "multiword_types.h"
#include "MW_target_hardware_resources.h"
#include "rt_nonfinite.h"
#include "rtGetInf.h"

/* Macros for accessing real-time model data structure */
#ifndef rtmGetBlkStateChangeFlag
# define rtmGetBlkStateChangeFlag(rtm) ((rtm)->blkStateChange)
#endif

#ifndef rtmSetBlkStateChangeFlag
# define rtmSetBlkStateChangeFlag(rtm, val) ((rtm)->blkStateChange = (val))
#endif

#ifndef rtmGetContStateDisabled
# define rtmGetContStateDisabled(rtm)  ((rtm)->contStateDisabled)
#endif

#ifndef rtmSetContStateDisabled
# define rtmSetContStateDisabled(rtm, val) ((rtm)->contStateDisabled = (val))
#endif

#ifndef rtmGetContStates
# define rtmGetContStates(rtm)         ((rtm)->contStates)
#endif
```

```
#ifndef rtmSetContStates
# define rtmSetContStates(rtm, val)    ((rtm)->contStates = (val))
#endif

#ifndef rtmGetDerivCacheNeedsReset
# define rtmGetDerivCacheNeedsReset(rtm) ((rtm)->derivCacheNeedsReset)
#endif

#ifndef rtmSetDerivCacheNeedsReset
# define rtmSetDerivCacheNeedsReset(rtm, val) ((rtm)->derivCacheNeedsReset = (val))
#endif

#ifndef rtmGetIntgData
# define rtmGetIntgData(rtm)           ((rtm)->intgData)
#endif

#ifndef rtmSetIntgData
# define rtmSetIntgData(rtm, val)      ((rtm)->intgData = (val))
#endif

#ifndef rtmGetOdeDeltaY
# define rtmGetOdeDeltaY(rtm)          ((rtm)->OdeDeltaY)
#endif

#ifndef rtmSetOdeDeltaY
# define rtmSetOdeDeltaY(rtm, val)     ((rtm)->OdeDeltaY = (val))
#endif

#ifndef rtmGetOdeF
# define rtmGetOdeF(rtm)               ((rtm)->odeF)
#endif

#ifndef rtmSetOdeF
# define rtmSetOdeF(rtm, val)          ((rtm)->odeF = (val))
#endif

#ifndef rtmGetOdeX0
# define rtmGetOdeX0(rtm)              ((rtm)->odeX0)
#endif

#ifndef rtmSetOdeX0
# define rtmSetOdeX0(rtm, val)         ((rtm)->odeX0 = (val))
#endif

#ifndef rtmGetPeriodicContStateIndices
# define rtmGetPeriodicContStateIndices(rtm) ((rtm)->periodicContStateIndices)
#endif

#ifndef rtmSetPeriodicContStateIndices
# define rtmSetPeriodicContStateIndices(rtm, val) ((rtm)->periodicContStateIndices =
(val))
#endif

#ifndef rtmGetPeriodicContStateRanges
# define rtmGetPeriodicContStateRanges(rtm) ((rtm)->periodicContStateRanges)
#endif

#ifndef rtmSetPeriodicContStateRanges
# define rtmSetPeriodicContStateRanges(rtm, val) ((rtm)->periodicContStateRanges =
(val))
#endif

#ifndef rtmGetZCCacheNeedsReset
# define rtmGetZCCacheNeedsReset(rtm)  ((rtm)->zCCacheNeedsReset)
```

```
#endif

#ifndef rtmSetZCCacheNeedsReset
# define rtmSetZCCacheNeedsReset(rtm, val) ((rtm)->zCCacheNeedsReset = (val))
#endif

#ifndef rtmGetdX
# define rtmGetdX(rtm)                 ((rtm)->derivs)
#endif

#ifndef rtmSetdX
# define rtmSetdX(rtm, val)            ((rtm)->derivs = (val))
#endif

#ifndef rtmGetErrorStatus
# define rtmGetErrorStatus(rtm)        ((rtm)->errorStatus)
#endif

#ifndef rtmSetErrorStatus
# define rtmSetErrorStatus(rtm, val)   ((rtm)->errorStatus = (val))
#endif

#ifndef rtmGetStopRequested
# define rtmGetStopRequested(rtm)      ((rtm)->Timing.stopRequestedFlag)
#endif

#ifndef rtmSetStopRequested
# define rtmSetStopRequested(rtm, val) ((rtm)->Timing.stopRequestedFlag = (val))
#endif

#ifndef rtmGetStopRequestedPtr
# define rtmGetStopRequestedPtr(rtm)   (&((rtm)->Timing.stopRequestedFlag))
#endif

#ifndef rtmGetT
# define rtmGetT(rtm)                  (rtmGetTPtr((rtm))[0])
#endif

/* Block signals (auto storage) */
typedef struct {
  real_T L[4];
  real_T PNew[4];                      /* '<S7>/RLS' */
  real_T TmpSignalConversionAtSFunct[2];/* '<S7>/RLS' */
  real_T thetaNew[2];                  /* '<S7>/RLS' */
  real_T FilterCoefficient;            /* '<S5>/Filter Coefficient' */
  real_T Memory;                       /* '<Root>/Memory' */
  real_T IntegralGain;                 /* '<S5>/Integral Gain' */
  real_T Gain;                         /* '<S11>/Gain' */
  real_T sqrtA0timesA1;
  real_T DataTypeConversion2;          /* '<Root>/Data Type Conversion2' */
  real_T UnitConversion;               /* '<S1>/Unit Conversion' */
  real_T Rollover_Index_Estimation;    /* '<Root>/Product' */
  uint16_T Speed_Setpoint;             /* '<Root>/Speed Command' */
  uint16_T Switch;                     /* '<S8>/Switch' */
  uint8_T DataTypeConversion;          /* '<S6>/Data Type Conversion' */
} B_UGV_Arduino_code_T;

/* Block states (auto storage) for system '<Root>' */
typedef struct {
  real_T estimatedParameters[2];       /* '<S7>/delayTheta' */
  real_T P[4];                         /* '<S7>/delayL' */
  real_T UnitDelay_DSTATE;             /* '<S11>/Unit Delay' */
  real_T Memory_PreviousInput;         /* '<Root>/Memory' */
  struct {
    void *LoggedData;
  } Scope_PWORK;                       /* '<S11>/Scope' */

  uint8_T icLoad;                      /* '<S7>/delayTheta' */
  uint8_T icLoad_b;                    /* '<S7>/delayL' */
} DW_UGV_Arduino_code_T;

/* Continuous states (auto storage) */
typedef struct {
  real_T AnalogFilterDesign_CSTATE[3]; /* '<Root>/Analog Filter Design' */
  real_T Integrator_CSTATE;            /* '<S5>/Integrator' */
  real_T Filter_CSTATE;                /* '<S5>/Filter' */
} X_UGV_Arduino_code_T;

/* State derivatives (auto storage) */
typedef struct {
  real_T AnalogFilterDesign_CSTATE[3]; /* '<Root>/Analog Filter Design' */
  real_T Integrator_CSTATE;            /* '<S5>/Integrator' */
  real_T Filter_CSTATE;                /* '<S5>/Filter' */
} XDot_UGV_Arduino_code_T;

/* State disabled  */
typedef struct {
  boolean_T AnalogFilterDesign_CSTATE[3];/* '<Root>/Analog Filter Design' */
  boolean_T Integrator_CSTATE;         /* '<S5>/Integrator' */
  boolean_T Filter_CSTATE;             /* '<S5>/Filter' */
} XDis_UGV_Arduino_code_T;

/* Zero-crossing (trigger) state */
typedef struct {
  ZCSigState Encorderpulsestorevmintoms_Trig;/* '<S3>/Encorder pulses to rev//min to
m//s' */
} PrevZCX_UGV_Arduino_code_T;

#ifndef ODE8_INTG
#define ODE8_INTG

/* ODE8 Integration Data */
typedef struct {
  real_T *deltaY;                      /* output diff */
  real_T *f[13];                       /* derivatives */
  real_T *x0;                          /* Initial State */
} ODE8_IntgData;

#endif

/* Parameters (auto storage) */
struct P_UGV_Arduino_code_T_ {
  real_T PIDController2DOF1_D;         /* Mask Parameter: PIDController2DOF1_D
                                        * Referenced by: '<S5>/Derivative Gain'
                                        */
  real_T PIDController2DOF1_I;         /* Mask Parameter: PIDController2DOF1_I
                                        * Referenced by: '<S5>/Integral Gain'
                                        */
  real_T PIDController2DOF1_N;         /* Mask Parameter: PIDController2DOF1_N
                                        * Referenced by: '<S5>/Filter Coefficient'
                                        */
  real_T PIDController2DOF1_P;         /* Mask Parameter: PIDController2DOF1_P
                                        * Referenced by: '<S5>/Proportional Gain'
                                        */
  real_T PIDController2DOF1_b;         /* Mask Parameter: PIDController2DOF1_b
                                        * Referenced by: '<S5>/Setpoint Weighting
(Proportional)'
                                        */
  real_T PIDController2DOF1_c;          /* Mask Parameter: PIDController2DOF1_c
                                         * Referenced by: '<S5>/Setpoint Weighting
(Derivative)'
                                         */
  uint32_T PWM_pinNumber;               /* Mask Parameter: PWM_pinNumber
                                         * Referenced by: '<S6>/PWM'
                                         */
  uint32_T ServoWrite_pinNumber;        /* Mask Parameter: ServoWrite_pinNumber
                                         * Referenced by: '<S9>/Servo Write'
                                         */
  real_T Constant6_Value;               /* Expression: 10
                                         * Referenced by: '<Root>/Constant6'
                                         */
  real_T Constant5_Value;               /* Expression: 10
                                         * Referenced by: '<Root>/Constant5'
                                         */
  real_T Gain1_Gain;                    /* Expression: 80
                                         * Referenced by: '<Root>/Gain1'
                                         */
  real_T Constant8_Value;               /* Expression: 0
                                         * Referenced by: '<Root>/Constant8'
                                         */
  real_T Gain2_Gain;                    /* Expression: -1.5
                                         * Referenced by: '<Root>/Gain2'
                                         */
  real_T Constant7_Value;               /* Expression: 11
                                         * Referenced by: '<Root>/Constant7'
                                         */
  real_T Out1_Y0;                       /* Computed Parameter: Out1_Y0
                                         * Referenced by: '<S11>/Out1'
                                         */
  real_T UnitDelay_InitialCondition;    /* Expression: 0
                                         * Referenced by: '<S11>/Unit Delay'
                                         */
  real_T Constant1_Value;               /* Expression: 60
                                         * Referenced by: '<S11>/Constant1'
                                         */
  real_T Radiusm_Value;                 /* Expression: 0.120
                                         * Referenced by: '<S11>/Radius(m)'
                                         */
  real_T Gain_Gain;                     /* Expression: 0.10472
                                         * Referenced by: '<S11>/Gain'
                                         */
  real_T AnalogFilterDesign_A[5];       /* Computed Parameter: AnalogFilterDesign_A
                                         * Referenced by: '<Root>/Analog Filter Design'
                                         */
  real_T AnalogFilterDesign_B;          /* Computed Parameter: AnalogFilterDesign_B
                                         * Referenced by: '<Root>/Analog Filter Design'
                                         */
  real_T AnalogFilterDesign_C;          /* Computed Parameter: AnalogFilterDesign_C
                                         * Referenced by: '<Root>/Analog Filter Design'
                                         */
  real_T AnalogFilterDesign_X0;         /* Expression: 0
                                         * Referenced by: '<Root>/Analog Filter Design'
                                         */
  real_T Constant9_Value;               /* Expression: 0
                                         * Referenced by: '<Root>/Constant9'
                                         */
  real_T Integrator_IC;                 /* Expression: InitialConditionForIntegrator
                                         * Referenced by: '<S5>/Integrator'
                                         */
  real_T Filter_IC;                     /* Expression: InitialConditionForFilter
                                         * Referenced by: '<S5>/Filter'
                                         */
  real_T Constant3_Value;              /* Expression: 0.312
                                         * Referenced by: '<Root>/Constant3'
                                         */
  real_T g_Gain;                       /* Expression: 9.81
                                         * Referenced by: '<Root>/g'
                                         */
  real_T Constant1_Value_f;            /* Expression: 0.255
                                         * Referenced by: '<Root>/Constant1'
                                         */
  real_T UGV_Height_Value;             /* Expression: 0
                                         * Referenced by: '<Root>/UGV_Height '
                                         */
  real_T Gain_Gain_i;                  /* Expression: 2
                                         * Referenced by: '<Root>/Gain'
                                         */
  real_T Constant2_Value;              /* Expression: -9.81
                                         * Referenced by: '<Root>/Constant2'
                                         */
  real_T Constant1_Value_b;            /* Expression: 6.276
                                         * Referenced by: '<S10>/Constant1'
                                         */
  real_T Enable_Value;                 /* Expression: 1
                                         * Referenced by: '<S7>/Enable'
                                         */
  real_T AdaptationParameter1_Value[4];/* Expression: initializationParams.adg1
                                         * Referenced by: '<S7>/AdaptationParameter1'
                                         */
  real_T AdaptationParameter2_Value;   /* Expression: initializationParams.adg2
                                         * Referenced by: '<S7>/AdaptationParameter2'
                                         */
  real_T InitialParameters_Value[2];   /* Expression: initializationParams.theta0
                                         * Referenced by: '<S7>/InitialParameters'
                                         */
  real_T InitialCovariance_Value[4];   /* Expression: initializationParams.L0
                                         * Referenced by: '<S7>/InitialCovariance'
                                         */
  real_T DesiredRollAngle_Threshold;   /* Expression: 5
                                         * Referenced by: '<Root>/Desired Roll Angle'
                                         */
  real_T Gain1_Gain_l;                 /* Expression: pi/180
                                         * Referenced by: '<S4>/Gain1'
                                         */
  real_T Memory_X0;                    /* Expression: 1.57
                                         * Referenced by: '<Root>/Memory'
                                         */
  real_T Switch1_Threshold;            /* Expression: 0.040
                                         * Referenced by: '<Root>/Switch1'
                                         */
  uint32_T SpeedCommand_p1;            /* Computed Parameter: SpeedCommand_p1
                                         * Referenced by: '<Root>/Speed Command'
                                         */
  uint32_T SpeedsensorInput_p1;        /* Computed Parameter: SpeedsensorInput_p1
                                         * Referenced by: '<Root>/Speed sensor Input'
                                         */
  uint32_T FromADXL337AccelarationSensor_p;/* Computed Parameter:
FromADXL337AccelarationSensor_p
                                            * Referenced by: '<Root>/From ADXL337

Accelaration Sensor '
                                            */
  uint16_T Bias1_Bias;                 /* Computed Parameter: Bias1_Bias
                                         * Referenced by: '<S10>/Bias1'
                                         */
  uint16_T delayTheta_DelayLength;     /* Computed Parameter: delayTheta_DelayLength
                                         * Referenced by: '<S7>/delayTheta'
                                        */
  uint16_T delayL_DelayLength;         /* Computed Parameter: delayL_DelayLength
                                        * Referenced by: '<S7>/delayL'
                                        */
  uint8_T ServoWrite_p1;               /* Computed Parameter: ServoWrite_p1
                                        * Referenced by: '<S9>/Servo Write'
                                        */
};

/* Real-time Model Data Structure */
struct tag_RTM_UGV_Arduino_code_T {
  const char_T *errorStatus;
  RTWSolverInfo solverInfo;
  X_UGV_Arduino_code_T *contStates;
  int_T *periodicContStateIndices;
  real_T *periodicContStateRanges;
  real_T *derivs;
  boolean_T *contStateDisabled;
  boolean_T zCCacheNeedsReset;
  boolean_T derivCacheNeedsReset;
  boolean_T blkStateChange;
  real_T OdeDeltaY[5];
  real_T odeF[13][5];
  real_T odeX0[5];
  ODE8_IntgData intgData;

  /*
   * Sizes:
   * The following substructure contains sizes information
   * for many of the model attributes such as inputs, outputs,
   * dwork, sample times, etc.
   */
  struct {
    int_T numContStates;
    int_T numPeriodicContStates;
    int_T numSampTimes;
  } Sizes;

  /*
   * Timing:
   * The following substructure contains information regarding
   * the timing information for the model.
   */
  struct {
    uint32_T clockTick0;
    uint32_T clockTickH0;
    time_T stepSize0;
    uint32_T clockTick1;
    uint32_T clockTickH1;
    SimTimeStep simTimeStep;
    boolean_T stopRequestedFlag;
    time_T *t;
    time_T tArray[2];
  } Timing;
};

/* Block parameters (auto storage) */
extern P_UGV_Arduino_code_T UGV_Arduino_code_P;

/* Block signals (auto storage) */
extern B_UGV_Arduino_code_T UGV_Arduino_code_B;

/* Continuous states (auto storage) */
extern X_UGV_Arduino_code_T UGV_Arduino_code_X;
```

```
/* Block states (auto storage) */
extern DW_UGV_Arduino_code_T UGV_Arduino_code_DW;

/* Model entry point functions */
extern void UGV_Arduino_code_initialize(void);
extern void UGV_Arduino_code_step(void);
extern void UGV_Arduino_code_terminate(void);

/* Real-time Model object */
extern RT_MODEL_UGV_Arduino_code_T *const UGV_Arduino_code_M;

/*-
 * These blocks were eliminated from the model due to optimizations:
 *
 * Block '<S16>/Output Dimension' : Unused code path elimination
 * Block '<S16>/Regressors Dimension' : Unused code path elimination
 * Block '<S16>/Sample Times and Data Type' : Unused code path elimination
 * Block '<S17>/Data Type Duplicate' : Unused code path elimination
 * Block '<S18>/Data Type Duplicate' : Unused code path elimination
 * Block '<S19>/Data Type Duplicate' : Unused code path elimination
 * Block '<S20>/Data Type Duplicate' : Unused code path elimination
 * Block '<S21>/Data Type Duplicate' : Unused code path elimination
 * Block '<S22>/Data Type Duplicate' : Unused code path elimination
 * Block '<S7>/Reset' : Unused code path elimination
 * Block '<S8>/Data Type Duplicate' : Unused code path elimination
 * Block '<S8>/Data Type Propagation' : Unused code path elimination
 * Block '<S2>/Data Type Conversion' : Eliminate redundant data type conversion
 * Block '<S3>/Data Type Conversion1' : Eliminate redundant data type conversion
 * Block '<S17>/Conversion' : Eliminate redundant data type conversion
 * Block '<S18>/Conversion' : Eliminate redundant data type conversion
 * Block '<S19>/Conversion' : Eliminate redundant data type conversion
 * Block '<S20>/Conversion' : Eliminate redundant data type conversion
 * Block '<S21>/Conversion' : Eliminate redundant data type conversion
 * Block '<S22>/Conversion' : Eliminate redundant data type conversion
 * Block '<S7>/ReshapeOutput' : Reshape block reduction
 * Block '<S7>/ReshapeRegressors' : Reshape block reduction
 */

/*-
 * The generated code includes comments that allow you to trace directly
 * back to the appropriate location in the model.  The basic format
 * is <system>/block_name, where system is the system number (uniquely
 * assigned by Simulink) and block_name is the name of the block.
 *
 * Use the MATLAB hilite_system command to trace the generated code back
 * to the model.  For example,
 *
 * hilite_system('<S3>')    - opens system 3
 * hilite_system('<S3>/Kp') - opens and selects block Kp which resides in S3
 *
 * Here is the system hierarchy for this model
 *
 * '<Root>' : 'UGV_Arduino_code'
 * '<S1>'   : 'UGV_Arduino_code/Angle Conversion'
 * '<S2>'   : 'UGV_Arduino_code/Data Convert'
 * '<S3>'   : 'UGV_Arduino_code/Data convert2'
 * '<S4>'   : 'UGV_Arduino_code/Degrees to Radians1'
 * '<S5>'   : 'UGV_Arduino_code/PID Controller (2DOF)1'
 * '<S6>'   : 'UGV_Arduino_code/PWM Motor Control'
 * '<S7>'   : 'UGV_Arduino_code/Recursive Least Squares Estimator'
 * '<S8>'   : 'UGV_Arduino_code/Saturation Dynamic'
 * '<S9>'   : 'UGV_Arduino_code/Servo Control'
 * '<S10>'  : 'UGV_Arduino_code/Data Convert/Conv to m//s2'
 * '<S11>'  : 'UGV_Arduino_code/Data convert2/Encorder pulses to rev//min to m//s'
 * '<S12>'  : 'UGV_Arduino_code/Recursive Least Squares Estimator/Check Enable Signal'
 * '<S13>'  : 'UGV_Arduino_code/Recursive Least Squares Estimator/Check Initial
Covariance'
 * '<S14>'  : 'UGV_Arduino_code/Recursive Least Squares Estimator/Check Initial
Parameters'
 * '<S15>'  : 'UGV_Arduino_code/Recursive Least Squares Estimator/Check Reset Signal'
 * '<S16>'  : 'UGV_Arduino_code/Recursive Least Squares Estimator/Check Signals'
 * '<S17>'  : 'UGV_Arduino_code/Recursive Least Squares Estimator/Data Type Conversion
Inherited0'
 * '<S18>'  : 'UGV_Arduino_code/Recursive Least Squares Estimator/Data Type Conversion
Inherited1'
 * '<S19>'  : 'UGV_Arduino_code/Recursive Least Squares Estimator/Data Type Conversion
Inherited2'
 * '<S20>'  : 'UGV_Arduino_code/Recursive Least Squares Estimator/Data Type Conversion
Inherited3'
 * '<S21>'  : 'UGV_Arduino_code/Recursive Least Squares Estimator/Data Type Conversion
Inherited4'
 * '<S22>'  : 'UGV_Arduino_code/Recursive Least Squares Estimator/Data Type Conversion
Inherited5'
 * '<S23>'  : 'UGV_Arduino_code/Recursive Least Squares
Estimator/MultiplyWithTranspose'
 * '<S24>'  : 'UGV_Arduino_code/Recursive Least Squares
Estimator/ProcessInitialCovariance'
 * '<S25>'  : 'UGV_Arduino_code/Recursive Least Squares
Estimator/ProcessInitialParameters'
 * '<S26>'  : 'UGV_Arduino_code/Recursive Least Squares Estimator/RLS'
 */
#endif                                 /* RTW_HEADER_UGV_Arduino_code_h_ */
```

```
/*
 * UGV_Arduino_code_private.h
 *
 * Student License - for use by students to meet course requirements and
 * perform academic research at degree granting institutions only.  Not
 * for government, commercial, or other organizational use.
 *
 * Code generation for model "UGV_Arduino_code".
 *
 * Model version              : 1.10
 * Simulink Coder version : 8.11 (R2016b) 25-Aug-2016
 * C source code generated on : Mon Mar 19 22:15:44 2018
 *
 * Target selection: ert.tlc
 * Note: GRT includes extra infrastructure and instrumentation for prototyping
 * Embedded hardware selection: Atmel->AVR
 * Code generation objectives: Unspecified
 * Validation result: Not run
 */

#ifndef RTW_HEADER_UGV_Arduino_code_private_h_
#define RTW_HEADER_UGV_Arduino_code_private_h_
#include "rtwtypes.h"
#include "multiword_types.h"

/* Private macros used by the generated code to access rtModel */
#ifndef rtmIsMajorTimeStep
# define rtmIsMajorTimeStep(rtm)       (((rtm)->Timing.simTimeStep) ==
MAJOR_TIME_STEP)
#endif

#ifndef rtmIsMinorTimeStep
# define rtmIsMinorTimeStep(rtm)       (((rtm)->Timing.simTimeStep) ==
MINOR_TIME_STEP)
#endif

#ifndef rtmGetTPtr
# define rtmGetTPtr(rtm)               ((rtm)->Timing.t)
#endif

#ifndef rtmSetTPtr
# define rtmSetTPtr(rtm, val)          ((rtm)->Timing.t = (val))
#endif

/* private model entry point functions */
extern void UGV_Arduino_code_derivatives(void);

#endif                                 /* RTW_HEADER_UGV_Arduino_code_private_h_ */
```