System Design, Modelling, and Control for an Off-Road Autonomous Ground Vehicle

John Mai

CMU-RI-TR-20-25

July 2020

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA, 15213

Thesis Committee:
George Kantor, Chair
David Wettergreen
Xuning Yang

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics

Abstract

Autonomy in passenger road vehicles has long been a goal for many research groups and companies, and there has been a significant amount of focus on achieving this endeavour. A lesser focused upon area is the task of precise autonomous driving in off-road environments, where widely varying terrain geometry & material as well as vehicle limitations impede modelling and control.

This thesis explores the process of designing and retrofitting a drive by wire system to an off-road passenger utility terrain vehicle, followed by modelling and control of the vehicle. In particular, we explore tractable methods of vehicle modelling in conditions where the internal combustion engine powered vehicle has difficulty following control inputs, such as low speed over rough terrain. We present a hybrid vehicle model which combines conventional actuator and motion models with a recurrent neural network learning residuals for the conventional models' predictions. This approach allows us to use simpler conventional model formulations while leveraging machine learning methods to correct approximations and assumptions using empirical data. Finally we present a method to plan kinematically feasible paths using Euler spirals to ensure curvature continuity, making use of power series approximations to reduce the computational cost of our overall solution.

Acknowledgments

I would like to thank my advisor, Mr. George Kantor, for guiding me throughout my journey at CMU. Without his advice, guidance, and support, I would not have been able to produce this work.
I would also like to thank all the members I've worked with on the Yamaha UTV project; in particular to Francisco Yandun, Srinivasan Vijayarangan, William Drozd, and Evan Cohen, with whom I spent countless hours out in the field testing this vehicle over scorching summer days through to freezing winter mornings.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

While the average passenger vehicle primarily spends its life on sealed roads, these types of road surfaces are only a subset of what most vehicles are capable of traversing. In rural or less developed areas, unpaved roads are often more prevalent, and vehicles travelling on these dirt or gravel trails experience a much different response from interacting with the terrain. In these conditions, approximations made by many simplified vehicle models such as a planar road surface and negligible wheel slip will not be as accurate, leading to less accurate model predictions. In order to account for these conditions, higher fidelity analytic 3D dynamic models have been formulated, accounting for 3D articulation and wheel-terrain interactions [12, 13, 14]. The issue with using these however is that complex analytical models can be very difficult to derive and calibrate. Applying these models requires knowledge of the geometry and dimensions of suspension and other drivetrain components, requiring either a CAD model or disassembly for inspection and measurements. Even after the model has been derived, there is still the task of calibrating the model parameters, as many parameters cannot be measured directly. For complex analytical models, this could require a large amount of instrumentation including on individual suspension linkages, wheels, and ground terrain material and geometry.

On top of interactions with the environment, the majority of passenger vehicles currently are driven by internal combustion engines, which are extremely complex systems. Analytical modelling of an internal combustion engine includes modelling fluid dynamics and combustion, making it cumbersome and prohibitively computationally expensive. Internal combustion engines are often coupled with a complex transmission system, due to their relatively small optimal speed range compared to electric motors. Depending on the type of transmission, this can include modelling fluid couplings or dynamically changing belt drive systems. The work required to model just the powertrain of an internal combustion often leads even vehicle manufacturers to opt for empirical models instead in their vehicle modelling work [15]. This can involve the use of an engine and/or wheel dynamometer, which can be expensive and time consuming to use.

For this project, a Yamaha UTV is required to autonomously traverse an off-road course, located at Gascola, in Penn Hills, PA. The vehicle will be aided by point clouds generated by an aerial vehicle flying overhead above the test site, and will use this data in conjunction with its onboard

sensors (camera, lidar, IMU) to assess traversability and navigate through the course to a goal location. At the goal location, a relatively level and open clearing, the vehicle will find a payload, a pallet with a footprint of about 1.0×1.3 m. The vehicle then need to use its onboard sensors to determine the payload's pose, then carefully maneuver itself in front of the pallet to pick it up. The payload docking procedure is similar to how a forklift picks up pallets, but this will be offroad on uneven terrain, using a vehicle that doesn't have the advantage of rear wheel steering.

The docking procedure will require careful planning in the limited space to ensure the final pose of the vehicle is in front of the pallet on the forkable side, oriented facing the pallet. As well as the nonholonomic constraints of the vehicle (we cannot turn on the spot or slide laterally at-will), we will also need to consider that the vehicle can only turn its steering wheel at a finite rate, and ensure that the path we plan has continuous curvature to account for this.

1.2 Overview

In the following chapters of this thesis, we'll review development and retrofitting of a new drive by wire system, before exploring methods of modelling and planning for the vehicle.

For the drive by wire system, we needed to retrofit enough actuators to the vehicle such that it can be driven entirely autonomously. As well as this, the coarse motion planner and vehicle controller we carried over from a previous phase of this project (MeshNav) drives the vehicle using speed and yaw rate commands, so we will need to develop controllers to interface these commands to actuator positions.

In the modelling chapter, we present a hybrid approach combining simple conventional models with machine learning methods. The goal of this is to avoid the difficult steps of formulating and calibrating complex analytical models, while still taking full advantage of the collected data. The conventional model can be used as a basis for a neural network, reducing the amount of training data and training time, while the neural network can be used to correct the approximations made by the conventional model.

Finally, we touch on the method we used for path planning and control of the vehicle. Our method takes waypoints generated by a conventional planner and applies a smoothing technique using Euler spirals to create a kinematically feasible path for the vehicle controller to track. Our implementation also makes use of power series approximations in order to reduce the computational cost of this smoothing method.

In summary, the primary contributions of this thesis are:

1. Present the drive by wire system we retrofitted to the UTV.
2. Propose a hybrid vehicle model predicting vehicle motion from raw actuator commands.
3. Propose a method of path planning using Euler spirals to create kinematically feasible paths.

Chapter 2

Systems Work

2.1 Introduction

We need a way to actuate all the necessary components in order to autonomously operate this vehicle. This could include the starter key, throttle, steering, brakes, and gear shifter. For the task at hand, once the vehicle was started and put in a "drive" gear, we found that controlling only the throttle and steering was sufficient for autonomous driving. The vehicle had an automatic transmission, and the engine braking was consistent and strong enough to slow the vehicle down without brakes. Thus we would first need to retrofit actuators on the steering wheel and throttle.

The coarse driving motion planner was carried over from the first stage of the project, and it controlled the vehicle with forward speed and yaw rate commands. Thus we would also need to develop controllers to interpret these commands and move the steering wheel and throttle appropriately.

Figure 2.1: Initial condition of the Yamaha UTV

2.2 Vehicle Overview

The vehicle used in this project is a 2014 model Yamaha Viking VI side-by-side UTV, with an automatic transmission. In its stock form, it is a 6-seater all-terrain vehicle with a wet weight of 1620 lb / 734 kg, although for the project the centre front and centre rear seats were removed to make room for electrical hardware. The vehicle also has a trailer bed in the rear which can tilt open to tip out its contents or expose the rear mounted engine and transmission. For the project, the trailer bed held four $LiFePO_4$ batteries each weighing approximately 20 kg, plus an onboard high power charger and power management unit. These provide electrical power the autonomy components, including computer, sensors, and actuators.

The vehicle was used previously used in 2014, where it was fitted with a Pronto4(TM) Series 4 System by Kairos Autonomi for drive by wire functionality. Onboard sensors include a NovAtel GPS, Xsens IMU, stereo camera, Velodyne 16, Velodyne 64, stock hall effect driveshaft speed sensor, and quadrature encoder sensors (attached to the front left wheel, front right wheel, and driveshaft).

2.3 Powertrain

This vehicle is powered by a liquid cooled 4-stroke, single cylinder, single overhead camshaft engine. Total engine displacement is 686 cm^3 [10, p. 2-2], and it produces a maximum of 34.5 kW[16].

The vehicle powertrain is controlled by a mechanical cable connecting the throttle pedal to a butterfly valve at the air intake. The vehicle features Yamaha's Ultramatic® transmission, a fully automatic CVT transmission. The driver can select from a low, high, or reverse gear, and rear wheel drive or four wheel drive.

2.3.1 Throttle

The original throttle actuator system by Kairos used a rotary actuator that pulls on a housed wire rope cable. This cable travels from a central control box housing all the actuators to the driver's footwell, where it pulls on the vehicle's throttle pedal [3]. From the pedal, this motion is transferred through the vehicle's housed throttle cable to the rear of the car where it moves a butterfly valve and throttle position sensor at the air intake.

For this project, the throttle valve is directly actuated to minimise backlash and play in the system.

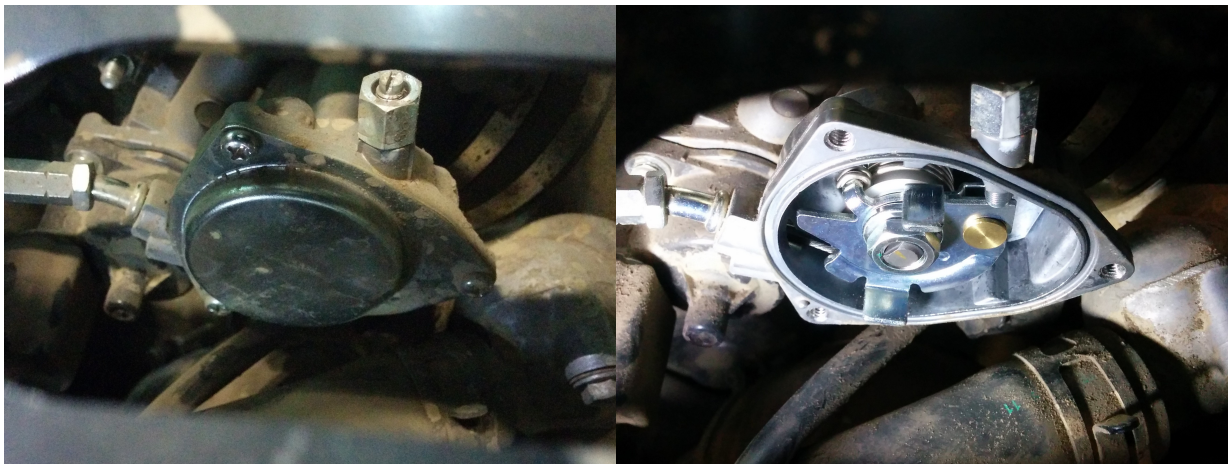


Figure 2.2: Bellcrank inside throttle body

Figure 2.2 shows the bellcrank connected to the butterfly valve, inside the throttle body located above the engine. A thin metal cover fastened by three M4x5.5 bolts protects the assembly from dust and water. This component converts the linear tension force of the throttle cable into rotational torque to open the butterfly valve controlling airflow. The butterfly valve is held close by a return spring, thus the force required to rotate the valve increases as the valve is opened. This was measured to be between 0.44 Nm and 0.62 Nm over the 85° range of motion.

A Dynamixel MX-28T servomotor was selected to actuate the throttle, which has a built in absolute encoder with a resolution of 0.088° (0.1% throttle range). For the torque requirements of the throttle, the servo operates near its peak efficiency (Figure 2.3) at a speed of about 35 RPM, theoretically going from $0-100\%$ throttle in under 150 ms

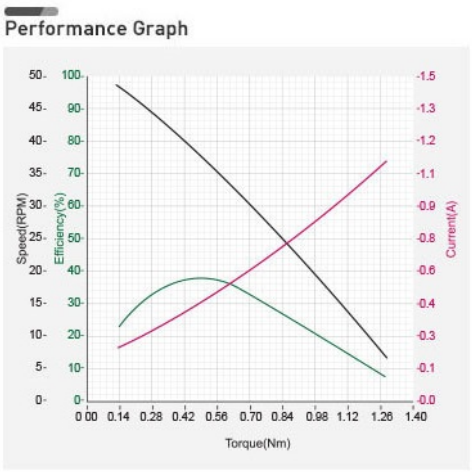


Figure 2.3: Performance graph for the MX-28 servomotor [1]

An adapter was designed in SolidWorks to mount the servo to the throttle body using the existing bolt holes for the bell crank cover (Figure 2.4).

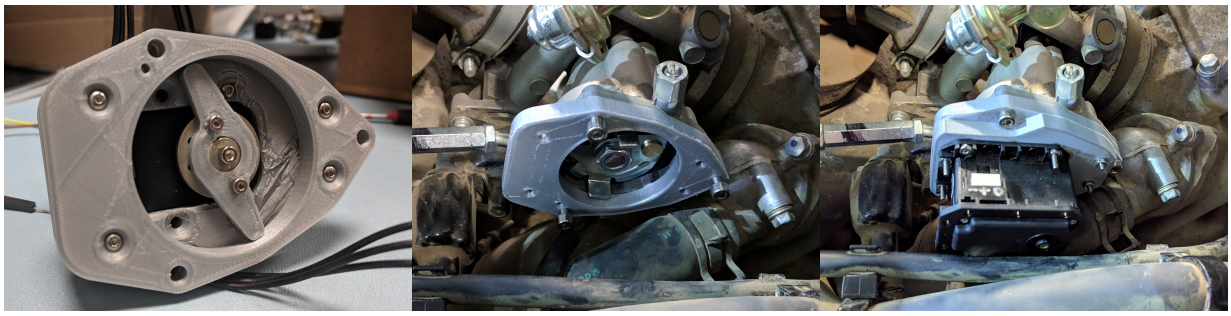


Figure 2.4: 3D-printed prototype throttle actuator mounting adapter

The adapter was designed in two pieces such that the final version could be milled out of 1/4” thick aluminium plate (Figure 2.5). The servo horn connects to a 3 mm thick aluminium piece with two arms that engage the prongs on the bellcrank, and pushes it clockwise to open the butterfly valve. The arms and the prongs are not fixed to each other, and the actuator relies on the return spring to close the valve. The advantage of this is that a driver is able to open the valve further using the throttle pedal, without affecting the servo motor.

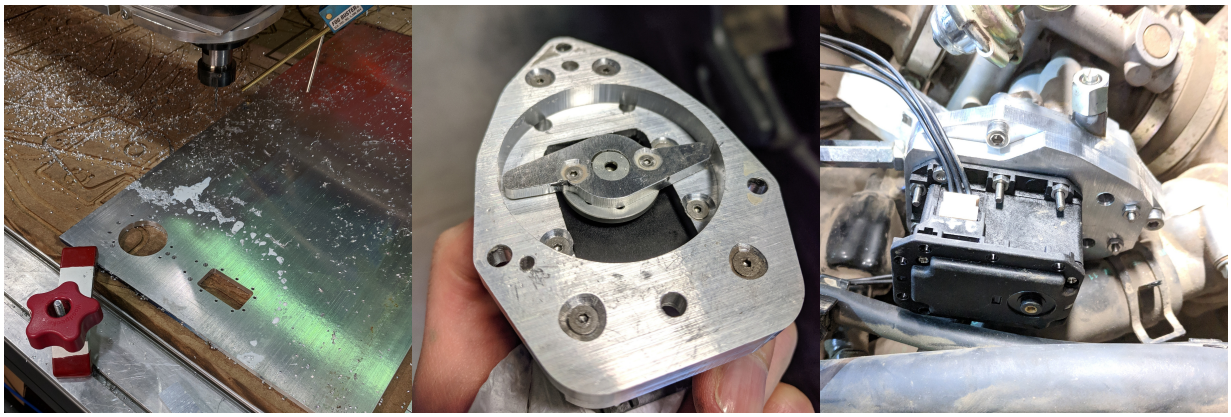


Figure 2.5: Aluminium throttle actuator mounting adapter

The MX-28T servo is powered by 12VDC from the vehicle’s onboard 12V DC-DC converters. It communicates with the main computer via TTL serial through a serial-USB converter, which is simply exposed as a serial port on the system.

The servo has an absolute encoder and is capable of continuous rotation. Care must be taken when positioning the zero-position of the servo, as the servo always rotates anticlockwise when resetting. If the servo is manually rotated anticlockwise even just a small amount past its zero-position and then reset, it does not rotate a short distance clockwise to return to zero, rather it rotates a near full 360° anticlockwise to return to zero. The return spring of the throttle valve has more than enough force to manually move the servo, and inertia will keep the servo rotating $10-20°$ beyond the throttle return stop, so the servo may be rotated past its zero-position if positioned too close to 0% vehicle throttle.

2.3.2 Speed control

Speed control on the vehicle is implemented using a PID controller with feed-forward and pitch compensation.

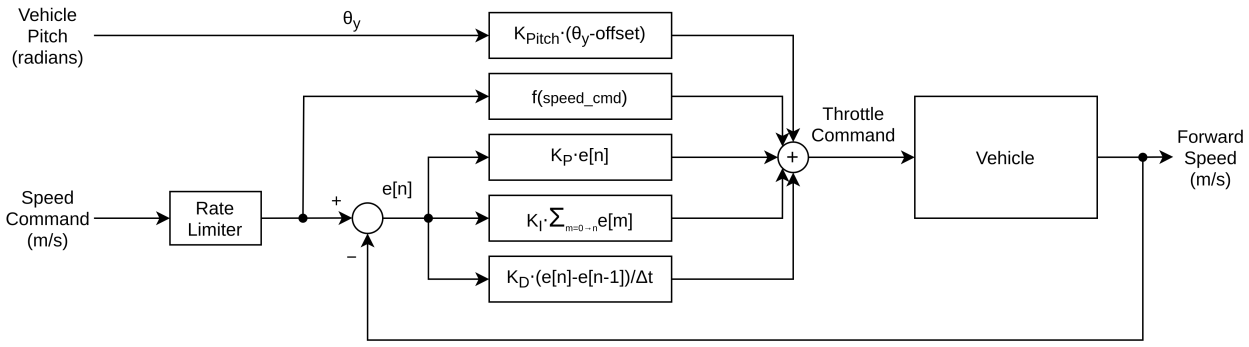


Figure 2.6: Block diagram of vehicle speed controller

The pitch compensation term adjusts the control output dependent on the current pitch of the vehicle. This increases the throttle when the vehicle is going uphill, and reduces the throttle when going downhill. The vehicle orientation is provided by the Xsens IMU, and a more positive pitch value indicates the vehicle is going downhill.

The feed-forward term reduces steady state error without requiring a higher integral gain which would result in sluggish controller behaviour. This is implemented by using a lookup table for the expected throttle command required for a given speed under steady state conditions on flat ground. Regions between measured data points are connected using linear interpolation. The response of the feed-forward term as a function of speed command is plotted below in Figure 2.7.

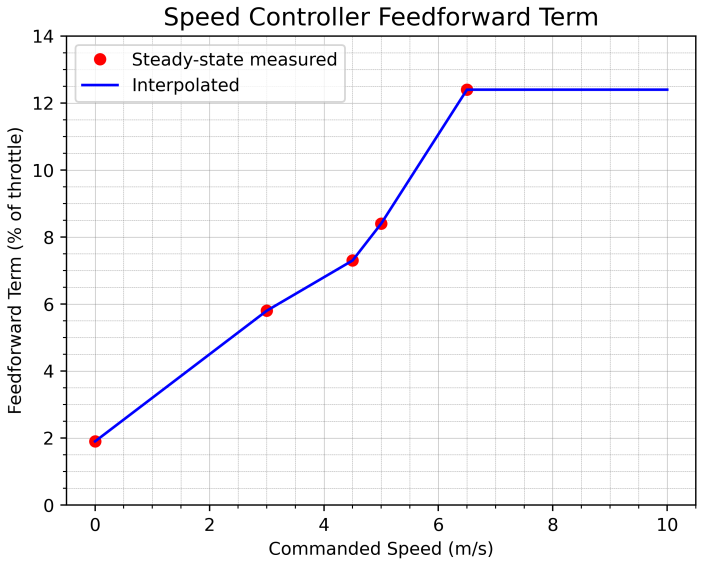


Figure 2.7: Feed-forward function plot

The controller also has a positive rate limit on the setpoint to smoothly accelerate the vehicle from standstill. When starting from standstill with a high velocity command and no rate limit, a large throttle command would be sent by the controller instantly which could lead to unsafe behaviour, particularly on narrow trails.

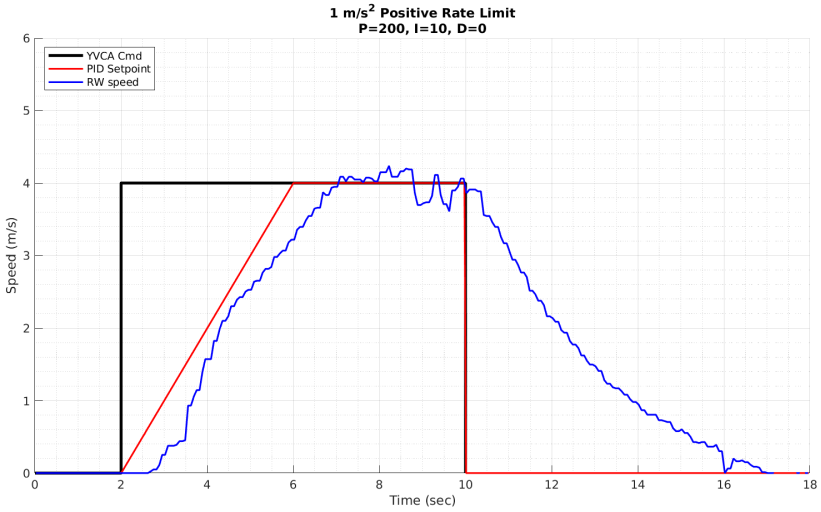


Figure 2.8: Positive rate limit on speed controller

The final tuned parameters used in the speed controller are outlined in the table below.

Parameter	**Value**	**Notes**
Control loop rate	100	Hz
Speed limit	6	m/s
Rate limit	0.01	0.01 m/s per control loop = 1 m/s^2
K_P	200	
K_I	10	
K_D	0	
K_{Pitch}	-1	Negative value because +ve pitch means vehicle is going downhill
Pitch offset	-2.9	
Control offset	2270	Servo encoder value for 0% throttle
Control range	950	Servo encoder value range for 0-100% throttle
Windup limit	$\pm$25	Maximum allowed integrated error

2.4 Steering

The Yamaha vehicle was fitted with a motor and chain driven steering ring by Kairos to turn the steering wheel.

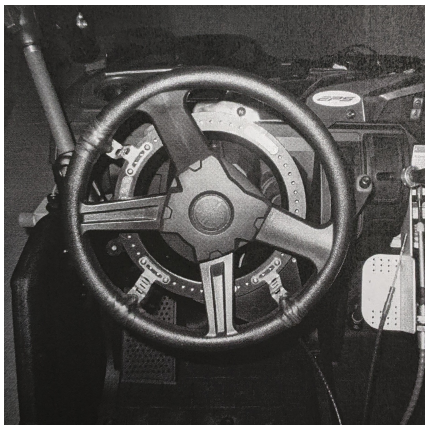


Figure 2.9: Steering actuator as fitted in 2014

Figure 2.10: Current state of steering actuator

The mechanism is made up of a large brushed DC motor which drives a ring attached to the steering wheel through a chain & sprocket linkage as shown below in Figure 2.11

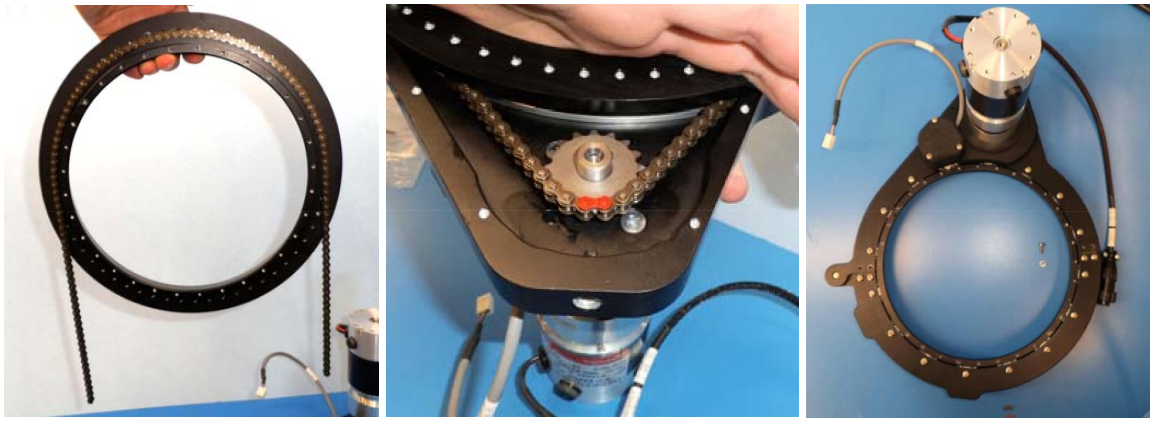


Figure 2.11: Steering actuator components [2]

This motor was originally driven by Kairos's included driver, but this was subsequently removed along with the rest of Kairos's control system in favour of direct control from the main computer. In its place, a Talon SRX motor controller is used, which features support for the encoder and has a builtin PID controller. This motor controller connects to the main computer via CAN through a CAN to USB converter. The setup allows for additional devices to be connected on the same CAN bus and communicate with the computer through the same adapter.

This motor controller is rated for 6-28V operation however [17], while the main 2P8S $LiFePO_4$ battery pack could reach a theoretical maximum of 28.8V when fully charged. While operating directly off the main battery would be preferred, to stay within the rated specifications the motor controller was driven from the 12V bus from the onboard DC-DC converters, which have a maximum output of 300W.
Testing this setup, we found that the motor would draw up to 20A from the 12V rail, shown below in Figure 2.12.

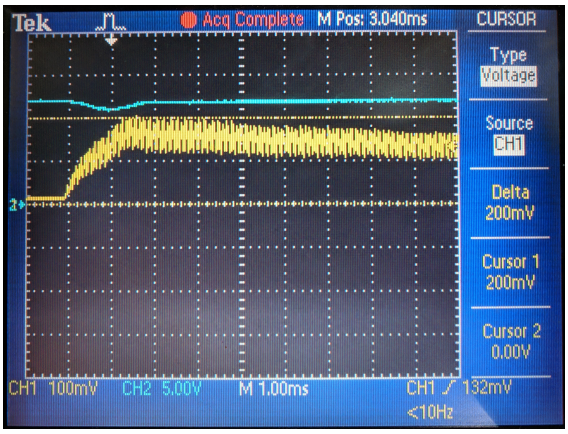


Figure 2.12: Current (CH1, yellow) & voltage (CH2, blue) at the input to the steering motor controller

In Figure 2.12, we can see that as the current draw (yellow) rises, the 12V rail voltage (blue) dips to 11V momentarily. However this dip occurs before the current has peaked, and recovers in 2 ms while the current draw is still around 20A peak. This suggests that the dip is not due to line losses, but due to converters having difficulty maintaining the 12V output during the very high $\frac{\Delta I}{\Delta t}$ event.

The peak draw of 20A @ 12V equates to 240W of power consumption, below the 300W maximum output of the converters. Aside from this motor, there aren't any other devices consuming a significant amount of power from the 12V rail, since the main computer runs off the 19V rail. However, sending large erratic commands to the motor controller would occasionally still cause the converters to shut down, despite the power draw remaining below the maximum output.

We noticed that the motor controller had very little input capacitance, and after some testing with a small 4080μF capacitor bank showed significant improvement in reliability, we installed the motor controller with a larger 33600μF capacitor bank, taking care to ensure the inrush current would not overwhelm the converters on startup. After the new capacitor bank was installed, the issue of the converters shutting down did not occur again. Figure 2.16 shows the motor controller installed with the capacitor bank and a relay to kill power to the steering motor for safety.

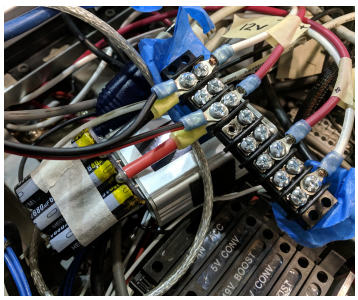


Figure 2.13: Initial test capacitor bank

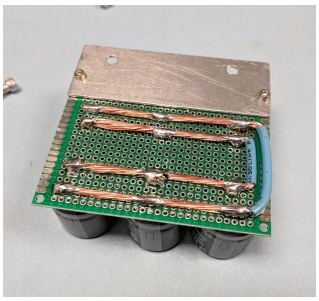


Figure 2.14: Final capacitor bank

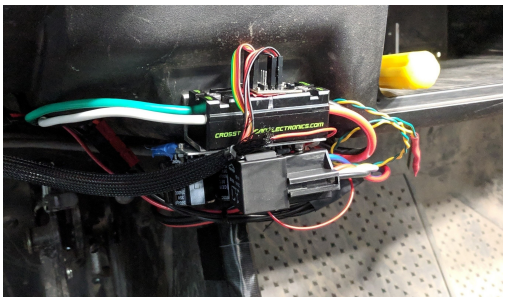


Figure 2.15: Installed motor controller with capacitor bank and kill relay

Figure 2.16: Steering motor controller

The behaviour of the steering wheel with this setup is shown below in Figures 2.17 and 2.18. Figure 2.17 shows behaviour when the I gain is too high. While the steering wheel behaves normally most of the time, when large alternating steering angle commands with short duration (black) are sent, the steering wheel angle (red) overshoots the expected behaviour (blue). Figure 2.18 shows corrected behaviour after tuning the PID parameters properly.

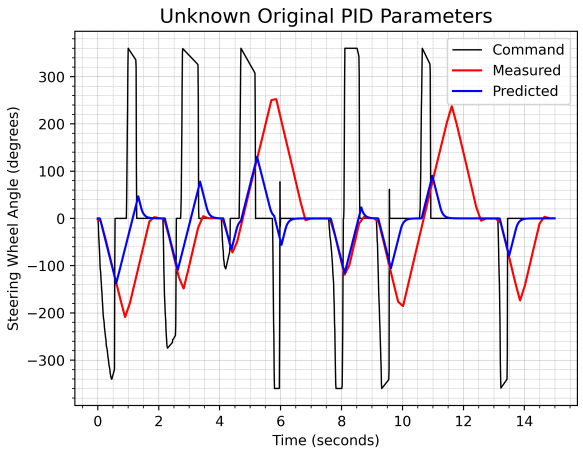


Figure 2.17: Steering wheel behaviour with original unknown parameters

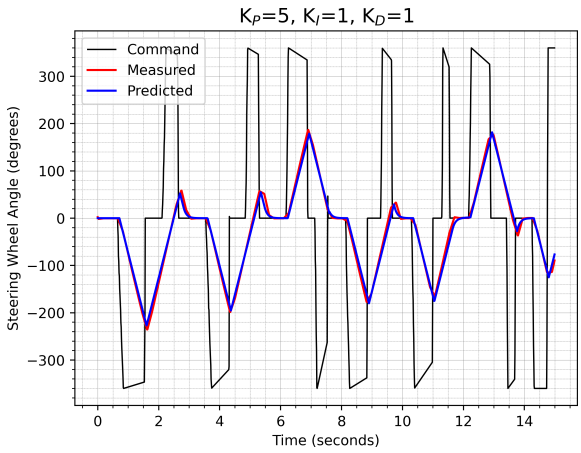


Figure 2.18: Steering wheel behaviour with new PID parameters

For high speed, coarse navigation, the vehicle uses the MeshNav planner/controller, which sends forward speed and yaw rate commands. While the speed commands can be read and used directly as the controller setpoint by the speed controller ROS node, an additional ROS node is used to read yaw rate commands and convert them to steering angle commands based on the current vehicle speed; steering angle affects yaw rate greater as the vehicle speed increases. This conversion is implemented with a simple kinematic bicycle model:

$$\text{steering\_angle} = \tan^{-1}\left(\frac{L \cdot \pi \cdot \text{yaw\_rate\_command}}{180 \cdot v}\right) \tag{2.1}$$

Where L is the vehicle wheelbase, v is the vehicle speed, and yaw_rate_command is given in degrees/second. steering_angle is the required angle that a theoretical wheel at the centre of the front axle would need to be steered to. A linear constant mapping is assumed between the front wheel steered angle and the steering wheel turned angle.

In order to ascertain this mapping between steering wheel angle and front wheel steered angle, we need to know the maximum angles the front wheels can be steered to, and the maximum angle amount that the steering wheel can be turned. The steering wheel actuator has a built in absolute encoder, so we can use the measurement from it: **452 degrees left** and **473 degrees right**.

Measuring the angles that the front wheels can be steered to is more difficult, as the steering mechanism doesn't just rotate the wheels about a vertical axis. Rather, the kingpin is at an angle, and this steering axis inclination results in the inside wheel pointing upwards and the outside wheel pointing downwards as the vehicle turns. Additionally, there is unknown underlying steering geometry which causes the inside wheel to turn more than the outside wheel; e.g. Ackerman steering geometry or similar.

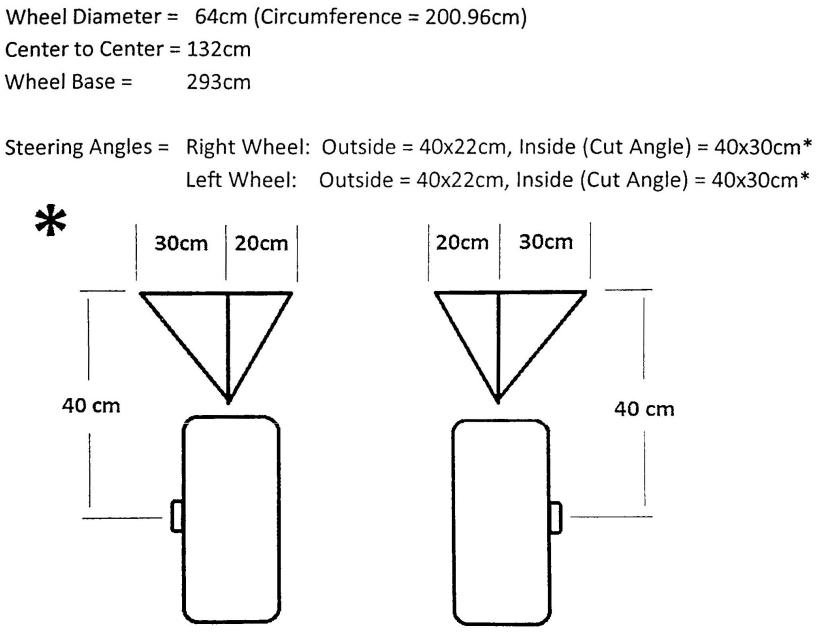


Figure 2.19: Specifications for wheel odometry measured by Kairos Autonomi [3]

Figure 2.19 shows front wheel steering angle measurements taken by Kairos Autonomi on this vehicle. Their measurements indicate a maximum steering angle of **36.9° out** and **26.6° in** on both front wheels. However there is no indication of what steering wheel angles were used in these tests, and no mention of the asymmetry between in the clockwise/anticlockwise turning limits of the steering wheel.

We also took measurements of our own by measuring the displacement of a pole attached to a front wheel as it is steered. A grid aligned to the vehicle body was marked out on the ground to align tape measures use to measure this displacement. This setup is shown below in Figure 2.20.

Figure 2.20: Setup used for measuring front wheel steering angle limits

For the front left wheel, the following measurements were obtained:

Steering wheel angle	Measured front wheel angle	Notes
$-360°$	$31.8°$	left (out)
$+360°$	$25.3°$	right (in)
$-473°$	$42.7°$	left limit (out)
$+452°$	$32.4°$	right limit (in)

While these results only offer two data points in each direction for one wheel, if we assumed a linear mapping between steering wheel angle and front wheel steered angle, we can get an idea of how linear the mapping is within the data points. Turning the steering wheel anticlockwise from a position of $-360°$ to the $-473°$ limit, we should expect to see a front wheel steered angle of $41.8°$ rather than the measured $42.7°$. To the right, from $+360°$ to the $+452°$ limit, we should expect an angle of $31.8°$ rather than the measured $32.4°$. This is an increase of 2.2% ($+0.9°$) and 1.9% ($+0.6°$) over the expected angle respectively. However to draw a strong conclusion more measurements would need to be taken over a range of steering angles and with a better setup. With the current setup, there was a significant amount of error that could be induced due to the flex of the measurement rod, and misalignment in the measurement lines. A measurement error of 1" for example affects the measured $32.4°$ value by $\pm 0.5°$.

2.5 Results

2.5.1 Lateral Control

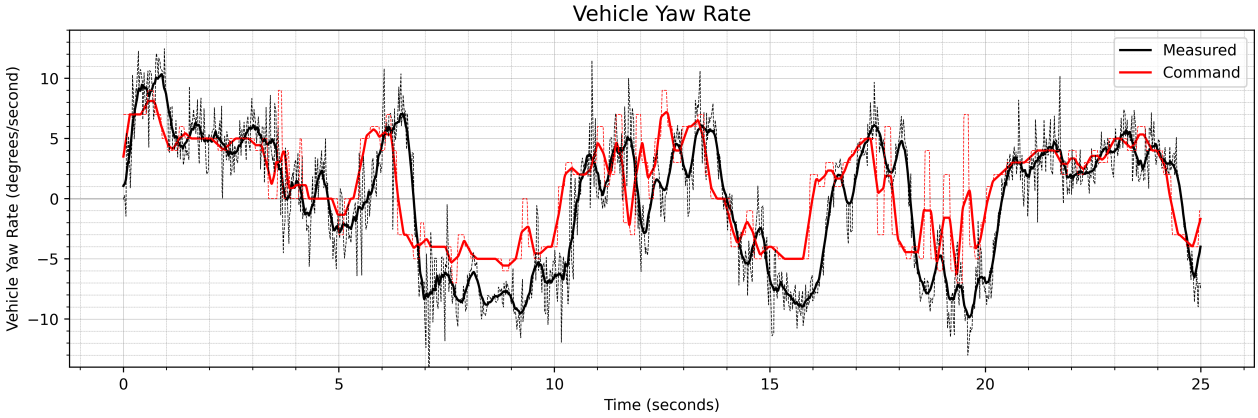


Figure 2.21: Vehicle yaw rates- raw (dashed) and 0.3 sec moving average (solid)

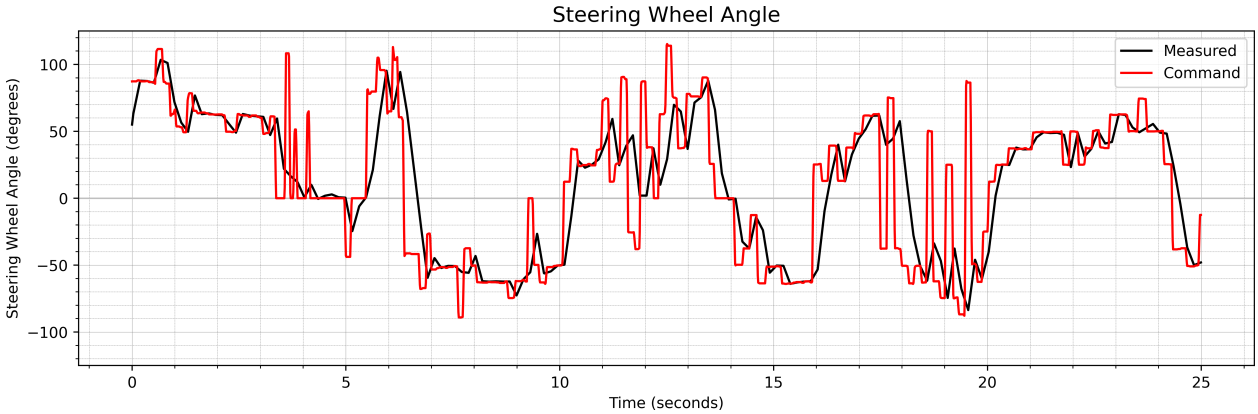


Figure 2.22: Vehicle steering wheel angles

Figures 2.21 & 2.22 show a 25 second excerpt of data logged from driving under MeshNav control at a speed of 5-6 m/s on level ground. Figure 2.21 shows the raw yawrates (dashed) along with the smoothed yawrates from a 0.3 second moving average. The measured yawrate (black) lags the commanded yawrate (red) by up to 0.5 seconds. The vehicle shows a tendency to yaw anticlockwise faster than commanded, as shown by the sections around $t = 8$, $t = 16$, and $t = 19$ where the measured yawrate has overshot the commanded yawrate. This is could be due to asymmetry in the steering mechanism; it was noted that the steering wheel has mechanical limits of 452° left and 473° right. The drive by wire system is limited to ±415°, and the steering was assumed to be symmetric and linear over this range.

Figure 2.22 shows steering angles over the same time period. The steering wheel has a response delay of about 50 ms, and a maximum turning rate of approximately 255 degrees/second. Over this excerpt, the steering wheel reaches a maximum angle of 100°, 24% of the limit.

2.5.2 Longitudinal Control

Figures 2.23 and 2.24 compare speed controller performance without and with a feedforward term respectively. For both runs the vehicle travels in a straight line on level ground.

Note that these tests were conducted prior to the introduction of the rate limiter, so the controller setpoint increases instantaneously. In both runs, there is a lag time of approximately 0.3 seconds between a speed command being issued and the vehicle beginning to respond.

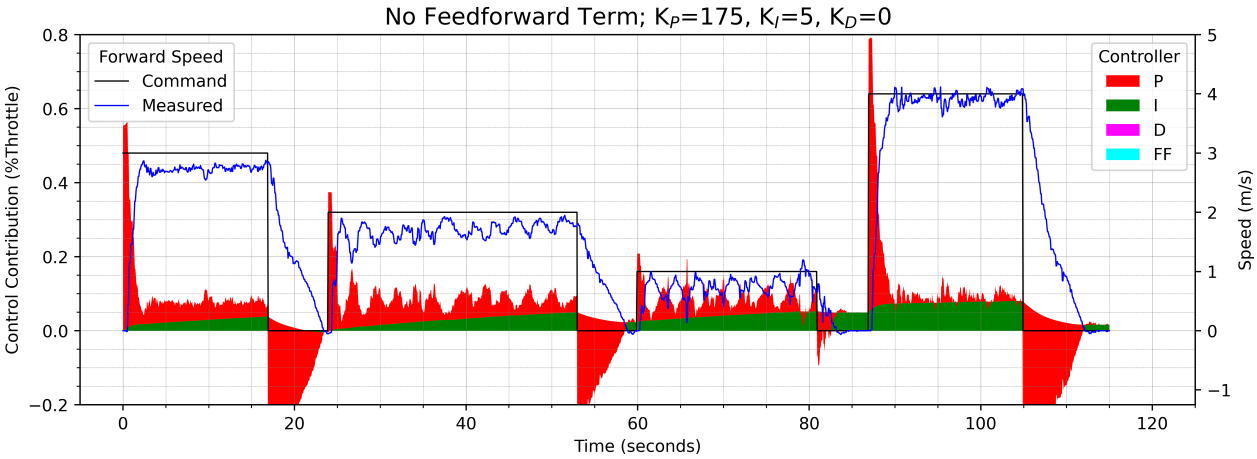


Figure 2.23: Vehicle longitudinal behaviour without feedforward term

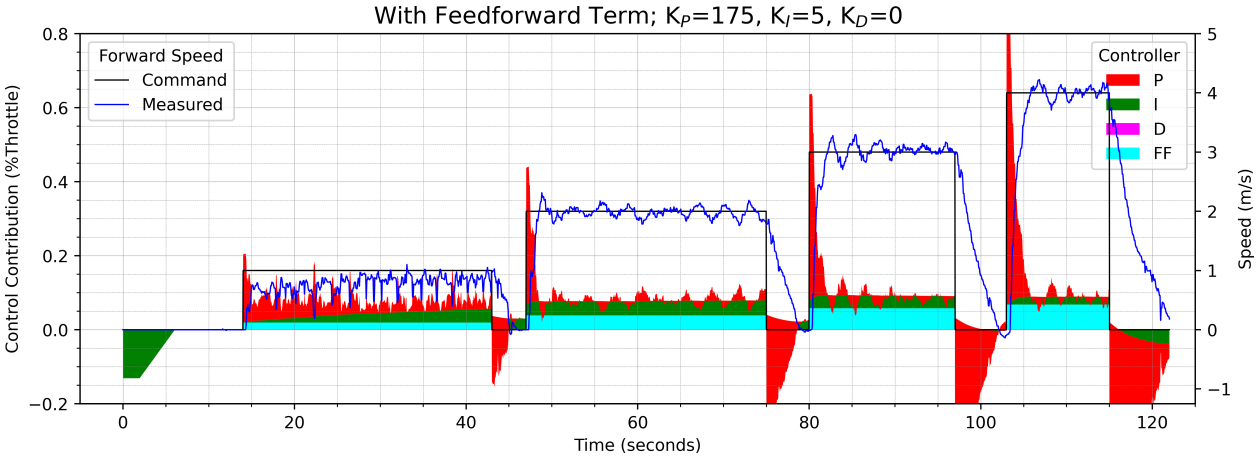


Figure 2.24: Vehicle longitudinal behaviour with feedforward term

In Figure 2.23, we can see that the controller without a feedforward term struggles to accurately track commands less than 4 m/s. For example, consider the 30 seconds plotted between $t = 23$ and $t = 54$ where the vehicle is given a 2 m/s speed command. The integral term (green) is constantly increasing as error builds up, but the proportional term (red) decreases as the measured speed (blue) approaches the commanded speed (black). The net effect is that the control signal (shown by the stacked area chart) stays roughly constant at 10% throttle, instead of increasing to eliminate the steady state error. In order to eliminate this error quicker, we could increase the

integral gain, however the overall controller would still perform poorly to transients, as we would be relying on accumulating error to reach the setpoint.

In Figure 2.24, we can see a 2 m/s command sent between $t = 46$ and $t = 75$. With the feedforward term (cyan) however, the vehicle speed is able to quickly reach the commanded speed. The aim of this additional term is to apply the correct throttle needed to maintain a commanded speed at steady state, leaving the other PID terms to handle transients. In the same figure, we can see that once the vehicle speed has reached the commanded speed, the majority of the control signal comes from the feedforward term, particularly for commanded speeds ≥ 2 m/s.

In both plots, the vehicle appears to have trouble maintaining a steady speed of 1 m/s. During testing it was found that maintaining this speed with a constant throttle command was not possible, and hence the feedforward term at this speed is interpolated, assuming at a feedforward term of 0% throttle results in 0 m/s. This was later increased, and the final feedforward function plotted in Figure 2.7 shows the function intersecting a 0 m/s command with a feedforward term of 2% throttle to improve low speed performance. This makes sense given the drivetrain system shown in detail in section 3.4.2. While 1-2% throttle is able to increase the engine speed a small amount, the engine speed would still be too low to engage the main clutch and hence no power can be transmitted from the engine to the wheels.

Figures 2.25 shows the behaviour of the speed controller starting on a slope. Figure 2.26 shows the pitch of the vehicle as the vehicle climbs the slope. The pitch has been inverted and adjusted so that 0° indicates level ground, and a positive pitch indicates the vehicle is driving uphill.

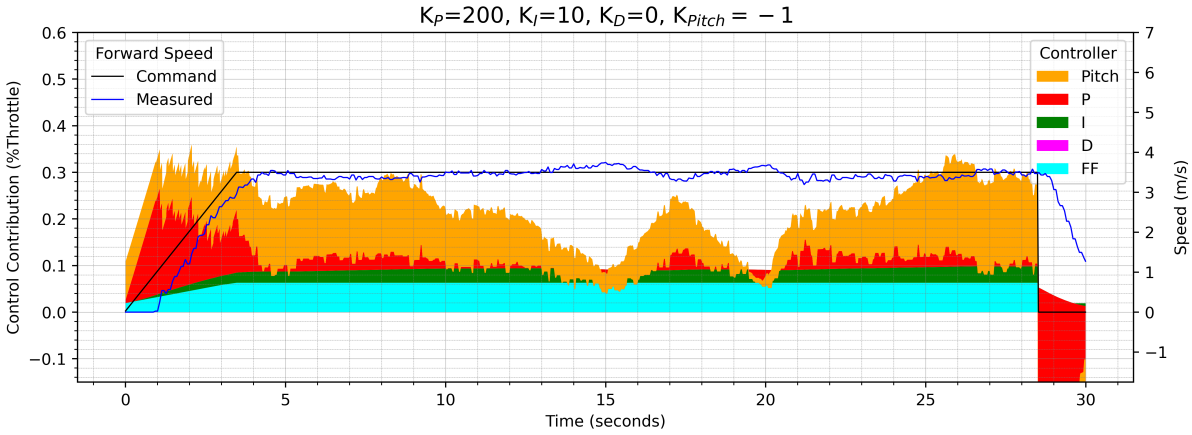


Figure 2.25: Vehicle longitudinal behaviour on a slope with controller pitch compensation

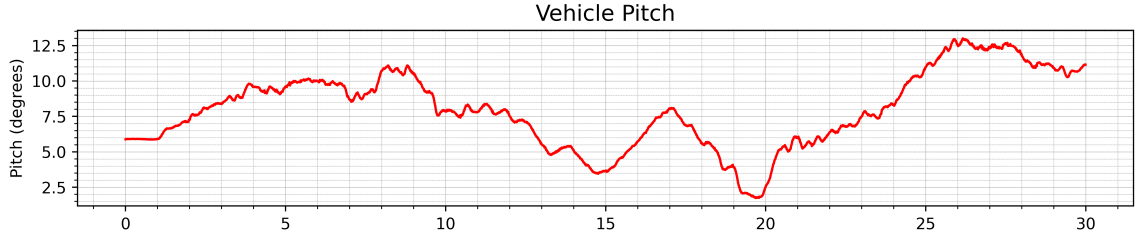


Figure 2.26: Pitch as the vehicle traverses the slope

From Figure 2.25 we can see that the majority of the control signal contribution comes from the pitch compensation term (orange).

There is a small delay at the start of the run, this is just from the driver holding down the brake to stop the vehicle rolling backwards down the slope before the autonomy is initialised.

Figure 2.27 shows the behaviour of the speed controller at low speed commands, under 2 m/s:

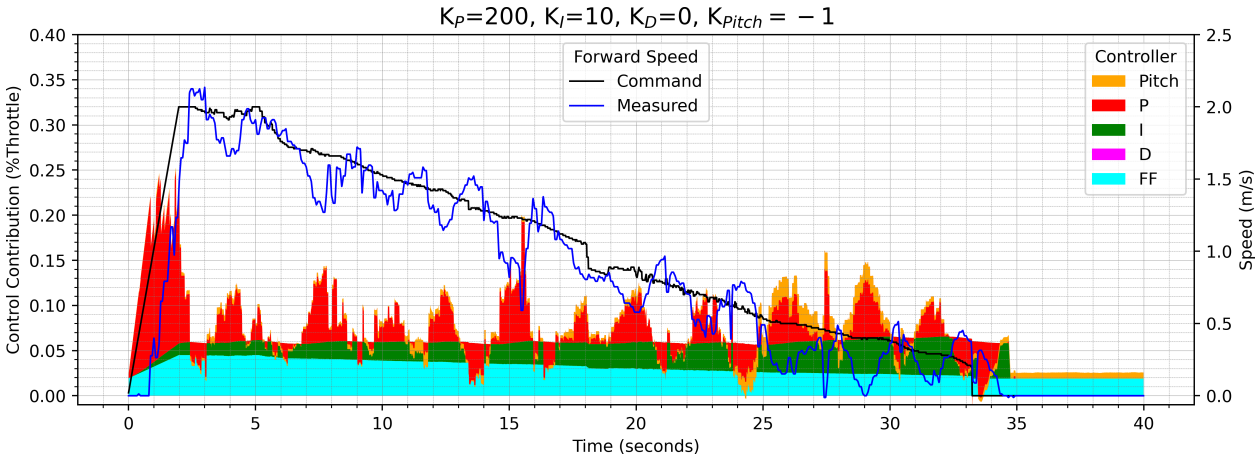


Figure 2.27: Behaviour of speed controller at low speed commands

This is from a precision navigation run, where the commanded speed is initially 2 m/s and gradually decreased as the vehicle approaches the goal position. There is significant overshooting of the setpoint, which gets worse as the commanded speed decreases. From around $t = 25s$ onwards, there is an increase in the contribution of the pitch compensation term (orange). This is in largely due to the pitching motion of the car as its forward velocity oscillates about the setpoint, rocking the occupants and IMU on its suspension.

Integrating measured and commanded speed over this range results in 31.6 m measured and 35.3 m commanded, an additional 11.7% that the controller had to apply in order to move the vehicle the correct distance to reach the goal.

2.6 Conclusion

In this chapter, we outlined the design decisions made while retrofitting the Yamaha UTV with actuators on the throttle valve and steering wheel for drive by wire operation. These actuators worked without any major issues over the 7 months that the vehicle was tested. Along with these, we also developed controllers for forward speed and yaw rate commands.

Upon evaluation of the yaw rate controller's performance, we found that it tended to cause the vehicle to yaw anticlockwise slightly faster than commanded, an issue which did not appear when commanding yaw rates clockwise. This is likely due to asymmetry in the steering mechanism (which was assumed to be symmetric), thus could be compensated for in our controller. The controller could also be improved by introducing feedback from either the Xsens IMU or the NovAtel GNSS/INS. This could correct any remaining error on the controller after it is tuned. Finally there is the issue of time delays up to approximately 0.5 seconds in the response

to commands. However this is likely due to the steering wheel's finite turning speed. It should be possible to increase this speed, either by powering the system from the onboard 19V rail or directly from the batteries. However, this would likely require upgrading the 19V rail's power capacity or swapping out the motor controller for one which can tolerate the full voltage range of the batteries respectively. Furthermore, the steering wheel turning rate is already much faster than what a human could achieve, and higher speeds may cause damage to the vehicle's stock electric power steering system, or pose a safety hazard to a human seated in the driver's seat.

Validation of the speed controller showed that it worked quite well despite rough uneven terrain; for example, Figure 2.25 shows at most a 5% deviation from the setpoint as the vehicle climbed a steep, rough slope. One shortcoming of the controller however, is its inability to keep a steady speed below 2 m/s on offroad terrain. We are unsure whether or not it is possible to achieve this feat, as even a human is unable to perform better than the current controller. Future work in this area could include a predictive term in the controller based on mapping local terrain geometry ahead of the vehicle, as this is one of the primary factors causing large errors about the speed setpoint at low speed.

Chapter 3

Vehicle Modelling

3.1 Introduction

For safe planning, control, and estimation of a vehicle, an accurate model predicting how the vehicle will respond to a given sequence of commands is necessary. However a vehicle is a complex mechanical system made up of many systems and components, including the steering mechanism, throttle actuator, suspension, engine, and drivetrain. One approach would be to consider each system and component individually to build a series of submodels which can be combined to map from input commands to output response. This can be extremely labour intensive, requiring disassembly of components for measurements not given in datasheets or service manuals, to near impossible due to complex dynamic interactions and proprietary components such as engine ECU logic. On top of this, field testing is still required in order to tune parameters that cannot be directly measured accurately. In order to simplify the process, approximate models are often used to describe multiple components in one, and automatic methods have been developed to tune parameters based on model error minimisation. In general these methods balance formulation and calibration complexity with model accuracy; more complex model formulations increase model calibration difficulty, but can lead to a more accurate model.

Alternative methods to building an analytical model are learned models, leveraging machine learning techniques to predict system response to input commands. The first immediate advantage of using a machine learning approach is not needing to trade off model and calibration complexity with model accuracy; with enough data, a more complex network can be trained without a proportional increase in labour intensity. Another advantage of a machine learning based approach is it makes no assumptions on how the vehicle will respond, and instead learns directly off the vehicle's actual response, whereas analytical models constrain the prediction of the vehicle to match the assumptions put in place during model formulation. The major downside to a machine-learning approach is the amount of data required to train the model. Vehicles are complex systems, so training a predictive model would require data demonstrating the behaviour of the vehicle over a large range of possible states, which can be a very difficult feat considering size of the state space for a vehicle.

By combining these two types of approaches, we can take advantage of the benefits of both; using approximations of the vehicle reduces the labour needed to formulate and tune an analytical

model, and this analytical model can be used as a basis for a machine learning based model. The final model will still have the flexibility of a model trained by machine learning based methods, but will require less data to train thanks to the underlying analytical model.

For consistency, the data we used to train and evaluate the following models contains only driving data following the finalisation of the vehicle's low level controllers. In total we have 2.59 hours of data contained in 181 separate bag files collected over a 1 month time period. These bags were categorised into high speed or low speed driving depending on whether or not the peak speed in the run exceeded 3 m/s. The categories were then split into train/validation/test sets. For the low speed category, we have 44, 11, 11 minutes of data split into train, validation, test respectively for a total of 65.4 minutes. For the high speed category, we have 65, 12, 13 minutes of data split into train, validation, test respectively for a total of 90.2 minutes. The trajectories of these datasets are overlayed and plotted below in Figures 3.1 and 3.2.

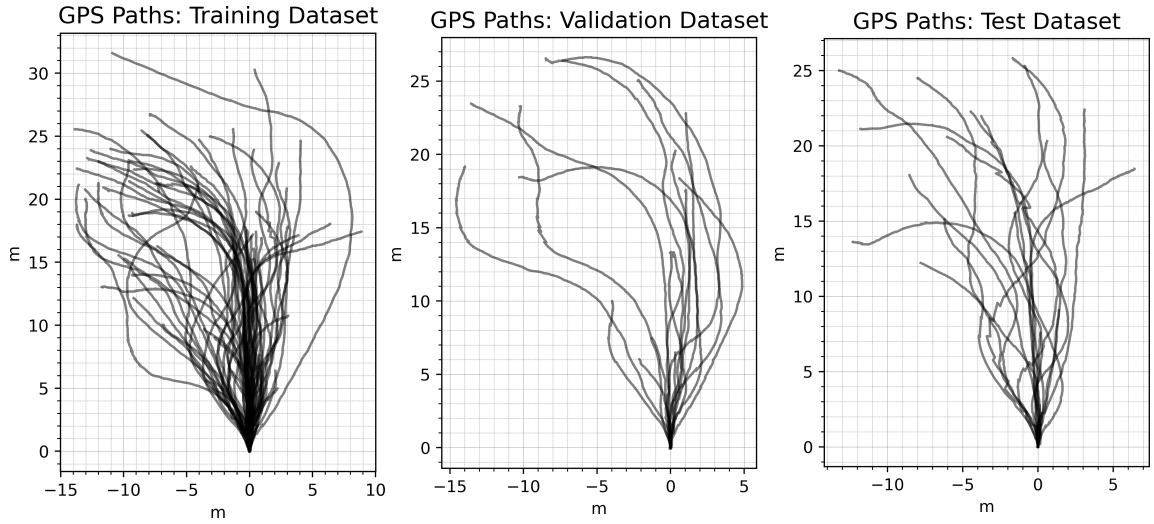


Figure 3.1: Summary of low speed dataset

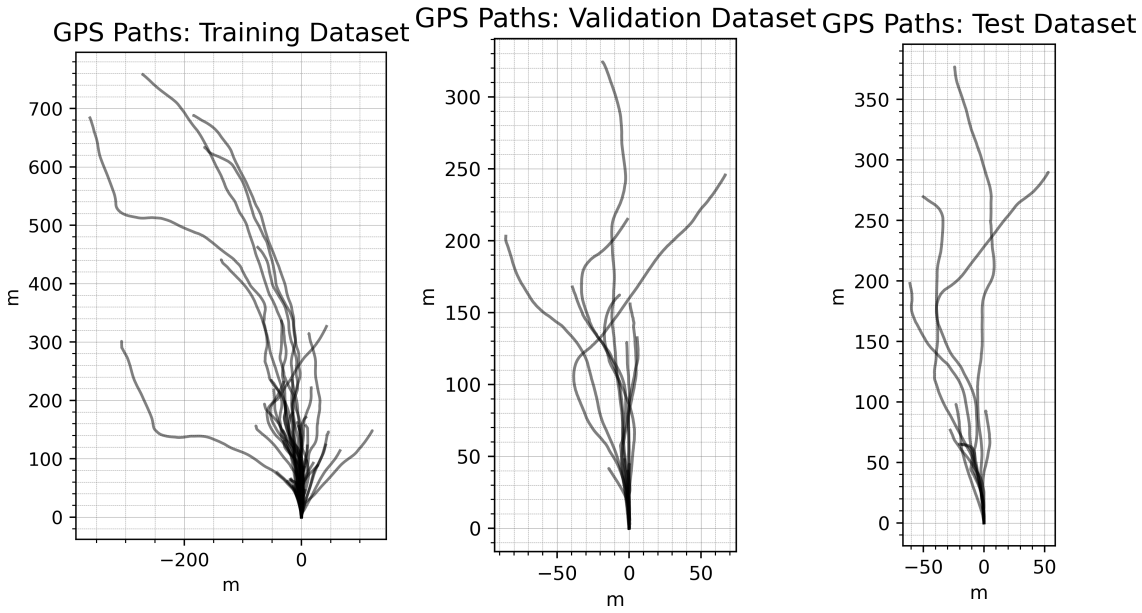


Figure 3.2: Summary of high speed dataset

From Figures 3.1 and 3.2 we can see that although we have more high speed driving data, it is in fewer bags containing long straight segments. This is because the higher speed data is mostly from coarse navigation through the long trails of the test site, whereas the low speed driving data is only from test runs of the local navigation system making significant adjustments to dock the vehicle to a payload. In this chapter, we are primarily interested in modelling the vehicle in the low speed operating region, as this is where we have the most varied data and where the vehicle operates while moving performing fine navigation to dock with the payload. Additionally, the vehicle speed controller has difficulty tracking the setpoint at low speed, so requires more work to model.

3.2 Related Work

Under certain conditions, namely at low speed over a flat surface, the motion of a front-wheel steered vehicle can be accurately modelled with a simple 2D kinematic bicycle model [18, p. 20-24], shown in Figure 3.12. This model assumes that the velocity vector at each bicycle wheel points in the direction the wheel is oriented, i.e. there is no lateral slip component at each wheel. This assumption is largely true for low acceleration maneuvers such as gentle low speed driving, but does not hold true where phenomena such as understeer, oversteer, or skidding occur. For these situations, this bicycle model can be extended to incorporate sideslip by considering lateral tyre forces, creating a dynamic bicycle model [18, p. 27-41].

Kong et al. [4] explore the use of kinematic and dynamic bicycle models for model-based control design and evaluate the prediction error of both models using experimental data of a passenger vehicle on asphalt averaging 13.8 m/s. Their dynamic bicycle model uses the assumption of a linear tyre model, where lateral tyre forces are directly proportional to slip angle. Their results showed that the dynamic bicycle model outperforms the kinematic bicycle model over four prediction steps with a 100 ms step size. However, they found that by using a larger step

size of 200 ms for the kinematic bicycle model, the prediction error is reduced and the model performs similarly well to their dynamic bicycle model.

Open-Loop Errors at Multiples of 200 ms (one step = 200 ms)

	Mean/Std Dev Errors at 200 ms steps			
Model and Discretization	1-Step	2-Step	3-Step	4-Step
Kinematic - 100 ms	0.08±0.04 m	0.16±0.06 m	0.27±0.11 m	0.40±0.19 m
Kinematic - 200 ms	0.07±0.04 m	0.14±0.05 m	0.22±0.08 m	0.33±0.13 m
Linear Tire - 100 ms	0.07±0.04 m	0.13±0.04 m	0.20±0.05 m	0.27±0.06 m
Linear Tire - 200 ms	0.08±0.04 m	0.15±0.05 m	0.23±0.06 m	0.31±0.08 m

Figure 3.3: Model prediction errors from the results of Kong et al.[4]

Seegmiller [13] expands upon 2D models, providing a general method to derive 3D velocity kinematics for any wheeled mobile robot joint configuration. His model formulation involves comprehensive modelling from the vehicle body, through suspension linkages, to individual wheels. This formulation takes into account 3D articulation, wheel-terrain interaction, and wheel liftoff. The benefit to this approach is that it is able to more accurately predict 6DoF vehicle state in extreme scenarios where previous models would fail, for example during aggressive driving, over steep and rough terrain, or while towing payloads.

The disadvantage to this approach is the extensive work required to build and calibrate the model. For example, suspension geometry and dimensions need to be known or estimated through experiments with instrumentation. While this is easier in the case of custom designed wheeled robot platforms with CAD models available, it can be difficult on full sized purchased vehicles, requiring digging through service manuals or disassembly of the actual vehicle. Other parameters, such as wheel ground contact angle, would require instrumented wheels to tune and validate.

The motion models discussed so far use vehicle speed and steering angle as inputs, but these are also state variables of the vehicle. In order to predict these to use in the motion model, we need actuator models estimating for example vehicle speed given a speed command to the controller.

Seegmiller's work uses powertrain models based on a first-order transient response with a time delay. These power train models predict wheel angular acceleration $\dot{\omega}$ given a commanded wheelspeed $\widetilde{\omega}$, and integrate to calculate wheelspeed ω. τ_c and τ_d are the time constant and time delay respectively [5].

$$\dot{\omega} = \frac{1}{\tau_c}\Big(\widetilde{\omega}\big(t-\tau_d\big) - \omega(t)\Big) \tag{3.1}$$

$$\omega(t) = \omega(t_0) + \int_{t_0}^{t} \frac{1}{\tau_c}\Big(\widetilde{\omega}\big(t-\tau_d\big) - \omega(t)\Big) d\tau \tag{3.2}$$

This model was used for robot vehicles including LandTamer, Zoë, Crusher, and RecBot [13]. Figure 3.4 compares the commanded, measured, and predicted wheelspeeds for the LandTamer skidsteer platform, under deceleration (left) and manual teleoperation (right):

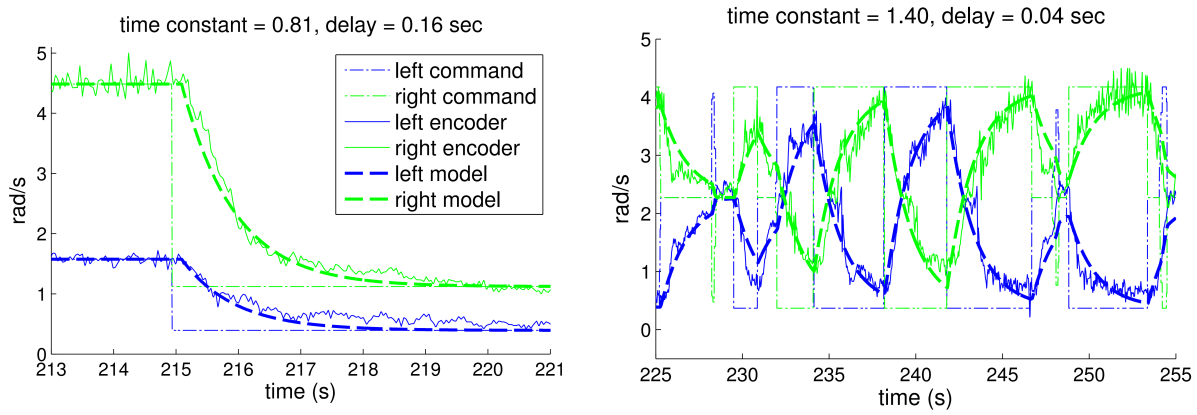


Figure 3.4: LandTamer wheelspeed modelling [5]

Alternate to conventional models, there has been work in predicting vehicle motion using machine learning based methods.

Bode [6] presents a data-driven approach for learning a forward predictive model using 7 hours of recorded data from the skid-steered Crusher UGV operating autonomously. This approach was motivated by the extreme driving conditions of the Crusher UGV, which traverses extreme terrain geometry with varying material, and experiences significant suspension travel and wheel slip even at low speed. His model uses a fully connected feed-forward neural network to predict along-track velocity (V_{AT}), cross-track velocity (V_{XT}) and rate of heading change (ω) given current velocities, orientations, and input commands to the motor controllers (Figure 3.5).

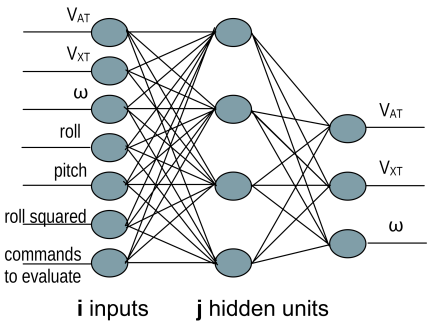


Figure 3.5: Structure of Bode's Neural Network Model [6]

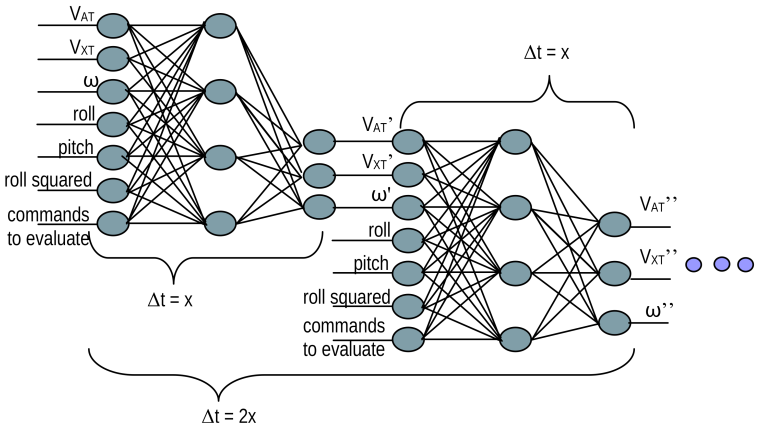


Figure 3.6: Chaining the neural network for prediction over multiple time steps [6]

To predict over multiple time steps, the output velocities from a prediction can be fed as inputs to the same model architecture with input commands from the next time step. Vehicle orientation however (roll/pitch) cannot be estimated by the model, so are estimated using a ground plane estimate and attachment model [6, p. 13]. Cross validation showed that the most accurate predictions were produced using a time step of 0.25 seconds and 8 hidden units [6, p.14]. Finally

these output velocities can then be integrated to obtain an estimated position trajectory of the vehicle.

Liu [19] applies this model on the Yamaha Viking UTV, and evaluates it against kinematic and dynamic bicycle models. Test conditions are a fixed speed command of 3 m/s with trajectories over a duration of 20 seconds. His results (Figure 3.7) show that in many cases, the neural network model outperforms both other models. However in some cases, the dynamic bicycle model performs just as well even better than the the neural network model. The kinematic bicycle model however is less accurate, and in some cases drifts over 15 m away from the ground truth by the end of the trajectory.

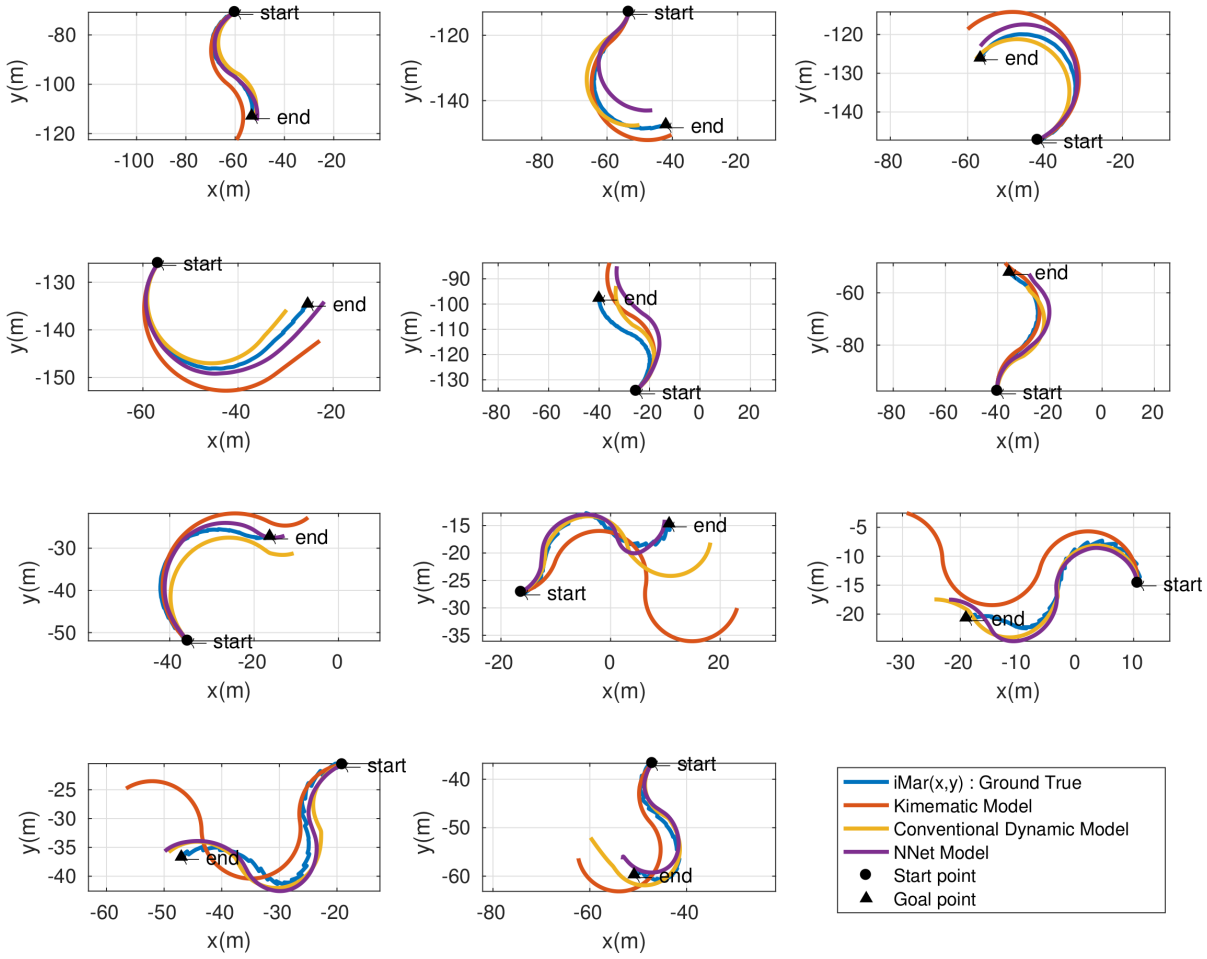


Figure 3.7: Comparison of vehicle models by Guan-Horng Liu [**?**]guanhorng

Finally, Ajay et al. [7] demonstrate the use of machine learning techniques applied in tandem with physics-based analytical models for modelling contact dynamics in ball bouncing and planar pushing scenarios. Their work is motivated by the complex behaviour of contacts and friction, which is undermodelled by conventional models. Five different approaches were compared, represented below in Figure 3.8:

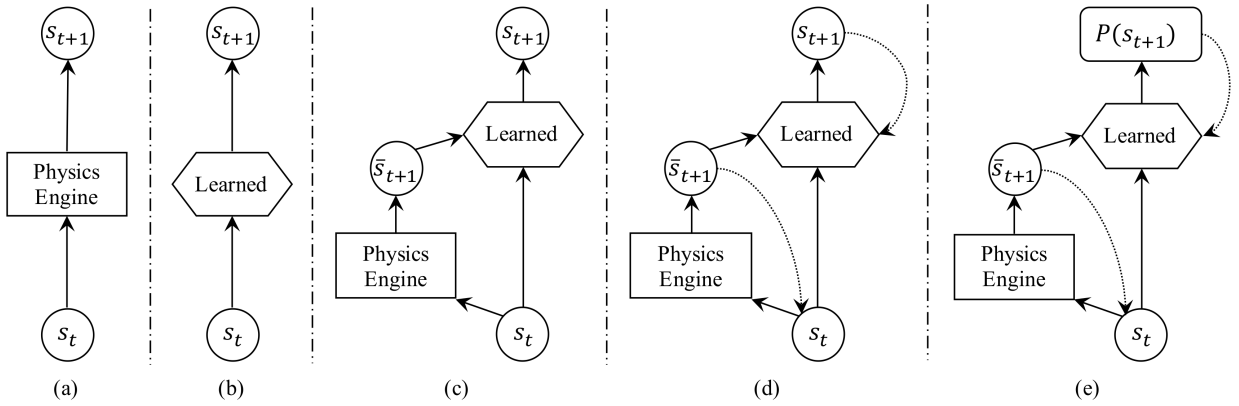


Figure 3.8: Five approaches to modelling contact dynamics by Ajay et al. [7]

Approaches (a) & (b) use only a physics based analytical model or a neural network respectively to predict next state given current state. Approach (c) combines the first two approaches, using both the current state and the prediction of the analytical model to give its own prediction. Approach (d) is similar, but replaces the feedforward neural network with a recurrent neural network. Approach (e) expands upon this by using a stochastic recurrent neural network to output a probability distribution for the next state. The results from this work show that the hybrid approaches (c/d/e) outperform the purely physics based (a) and purely data driven (b) approaches. Using recurrent architectures for the data driven block (d/e) was shown to correct for compounding error over long prediction horizons which occurred in approach (c) with a feedforward network architecture. Approach (e) offers the ability to also predict uncertainty, which is useful for their planning and control algorithms.

3.3 Conventional Motion Models

3.3.1 Kinematic Models

One method of modelling a vehicle is by considering the geometric constraints that define its motion, leading to a kinematic model. A kinematic vehicle model assumes there is no slip in the wheels, and often this can be sufficient to capture the behaviour of a vehicle at low speed where lateral accelerations are not significant. For a front wheel steered vehicle, a simple well-known model for this operating condition is the kinematic bicycle model, where the front wheels and rear wheels of a four-wheeled vehicle are combined to just a single front and rear wheel, as shown in Figure 3.12.

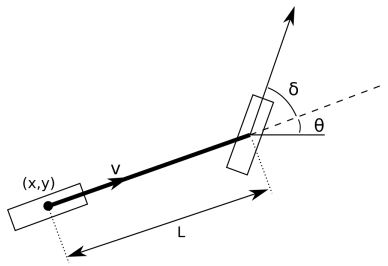


Figure 3.9: Rear axle reference

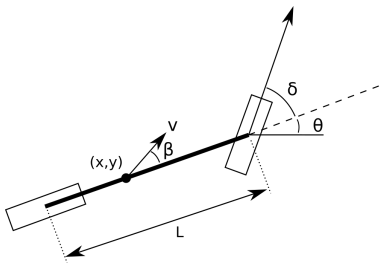


Figure 3.10: Centre of mass reference

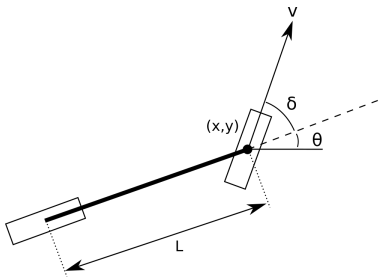


Figure 3.11: Front axle reference

Figure 3.12: Kinematic bicycle models with different reference points

The kinematic bicycle model is widely used because of its simplicity and adherence to the nonholonomic constraints of a car. The model considers the vehicle to be operating on a 2D plane, with its motion dependent solely on its forward velocity and the front wheel steer angle. The equations defining vehicle motion depend on the point of reference used; the centre of the rear axle, the centre of the front axle, or the vehicle's centre of mass [18]:

Rear axle reference:	Centre of mass reference:	Front axle reference:
$\dot{x} = v\cos\theta$	$\dot{x} = v\cos(\theta+\beta)$	$\dot{x} = v\cos(\theta+\delta)$
$\dot{y} = v\sin\theta$	$\dot{y} = v\sin(\theta+\beta)$	$\dot{y} = v\sin(\theta+\delta)$
$\dot{\theta} = \frac{v\tan\delta}{L}$	$\dot{\theta} = \frac{v\cos\beta\tan\delta}{L}$	$\dot{\theta} = \frac{v\sin\delta}{L}$
	$\beta = \tan^{-1}(\frac{L_r\tan\delta}{L})$	

These hard constraints rely on a "no-slip" condition assumption, where both wheels are assumed to be unable to slip in any direction.

3.3.2 Dynamic Models

While the no-slip asssumption is valid for low speeds, at higher speeds or low traction conditions lateral motion of the wheels can become non-negligible. In order to expand our model to take into account these motions, we have to consider the forces acting on the vehicle. A simple method is to expand upon the kinematic bicycle to include these forces, resulting in a dynamic bicycle model.

Consider the diagram in Figure 3.13. Without a no-slip assumption, the direction that a wheel is pointing is not necessarily the direction that the wheel travels in, and the angle between these two directions is the *slip angle*, α. This slip results in lateral forces being generated at the wheel, and the dynamic bicycle model considers these forces to determine the lateral and yaw accelerations of the vehicle.

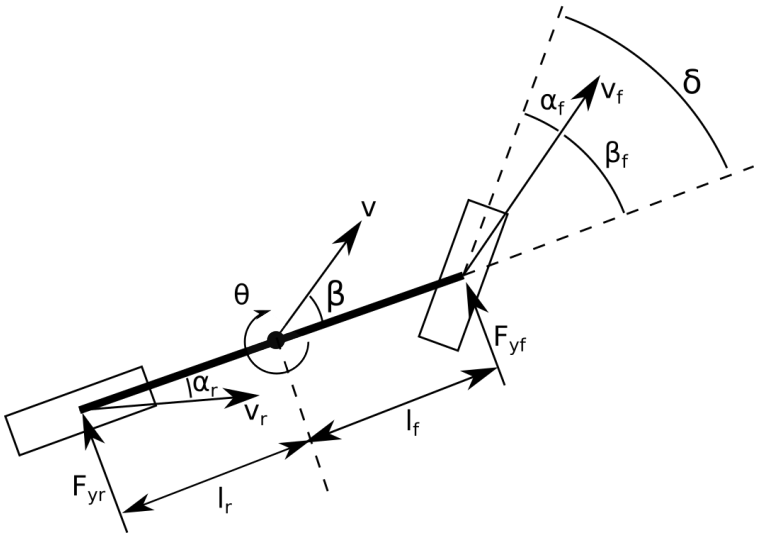


Figure 3.13: Dynamic bicycle model

Modelling lateral force (also known as cornering force) of the wheel can be difficult due to the compliant nature of tyres, and complicated interaction between tyres and different terrains. For a typical wheel, the relationship between lateral force and slip angle follows a similar trend to the graph in Figure 3.14. In the transitional region and beyond, behaviour is complicated and dependent on up to 20 parameters describing the tyre's contact patch and the forces and torques generate by it [20]. A widely used simplified tyre model is Pacejka's "Magic Formula" (Figure 3.15), named for its lack of physical basis. Despite this, the model fits a wide variety of tyres and conditions, and is parameterised by only four fitting coefficients, typically determined empirically [20].

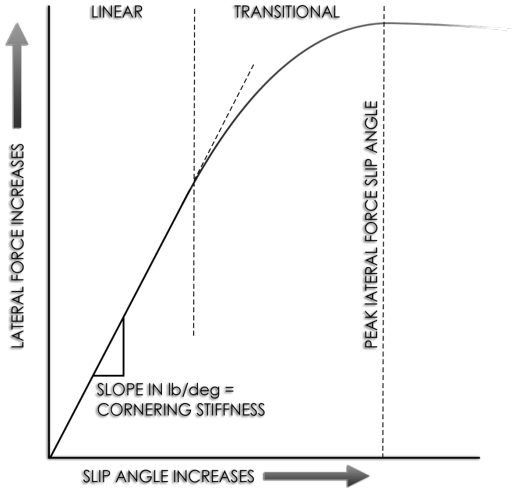


Figure 3.14: Typical lateral tyre force vs slip angle [8]

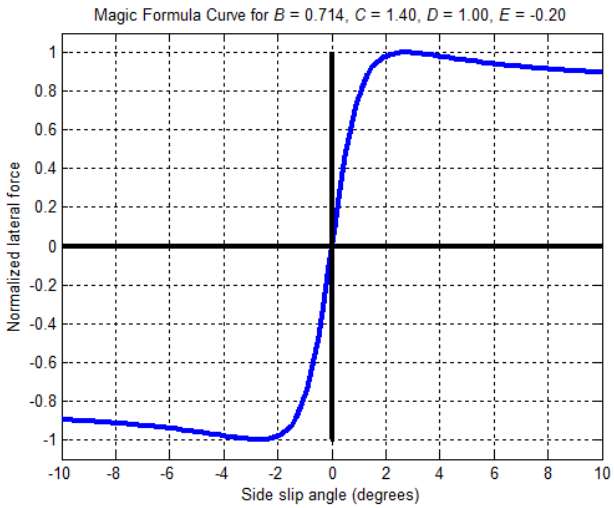


Figure 3.15: Pacejka Magic Formula Curve [9]

In our situation however, the vehicle will be moving at low speeds, so the slip angle was assumed

to be small enough for a linear tyre model to approximate:

$$F_{yf} = -C_f \alpha_f \qquad\qquad F_{yr} = -C_r \alpha_r$$

Where C_f and C_r are the lumped cornering stiffness coefficients for the front wheels and rear wheels.
Thus the overall lateral force F_y and yaw torque M_z acting on the vehicle are:

$$\begin{aligned} F_y &= F_{yf}\cos\delta + F_{xf}\cos(\frac{\pi}{2} - \delta) + F_{yr} \\ &= F_{yf}\cos\delta + F_{yr} && F_{xf} = 0 \text{ for rear wheel drive only vehicles} \\ &= F_{yf} + F_{yr} && \text{If small steering angle assumption is used} \end{aligned}$$

and

$$M_z = l_f F_{yf} - lr F_{yr}$$

In many texts ([21],[4]), a small steering angle assumption is used to simplify calculations, but in our usecase the vehicle will be moving at low speed with large steering angles, so this assumption is not used.
To calculate the lateral forces on each wheel, we need to calculate side-slip angles for the vehicle centre of mass (β) and each wheel (β_f,β_r) using the current longitudinal and lateral velocities of the centre of mass (v_x and v_y respectively).

$$\beta = \tan^{-1}\left(\frac{v_y}{v_x}\right) \qquad \beta_f = \tan^{-1}\left(\frac{v_{yf}}{v_{xf}}\right) = \tan^{-1}\left(\frac{v_y + l_f\dot{\theta}}{v_x}\right) \qquad \beta_r = \tan^{-1}\left(\frac{v_{yr}}{v_{xr}}\right) = \tan^{-1}\left(\frac{v_y - l_r\dot{\theta}}{v_x}\right)$$

The local side slip angles α can then be calculated, using the $\tan^{-1}(a) = a$ approximation from the small slip angle assumption:

$$\begin{aligned} \alpha_f &= \beta_f - \delta \\ &= \tan^{-1}\left(\frac{v_y + l_f\dot{\theta}}{v_x}\right) - \delta \\ &= \frac{v_y + l_f\dot{\theta}}{v_x} - \delta \\ &= \beta + \frac{l_f\dot{\theta}}{vx} - \delta \end{aligned} \qquad \begin{aligned} \alpha_r &= \beta_r \\ &= \tan^{-1}\left(\frac{v_y - l_r\dot{\theta}}{v_x}\right) \\ &= \frac{v_y - l_r\dot{\theta}}{v_x} \\ &= \beta - \frac{l_r\dot{\theta}}{vx} \end{aligned}$$

Finally the overall lateral force F_y and yaw torque M_z acting on the vehicle can be defined in

terms of slip angle:

$$\begin{aligned} F_y &= F_{yf}\cos\delta + F_{yr} \\ &= -C_f\cos(\delta)\alpha_f - C_r\alpha_r \\ &= -C_f\cos(\delta)\Big(\beta + \frac{v_y + l_f\dot{\theta}}{v_x} - \delta\Big) - C_r\Big(\beta - \frac{l_r\dot{\theta}}{v_x}\Big) \end{aligned}$$

and

$$\begin{aligned} M_z &= l_f F_{yf} - lrF_{yr} \\ &= -l_f C_f\Big(\beta + \frac{v_y + l_f\dot{\theta}}{v_x} - \delta\Big) + l_r C_r\Big(\beta - \frac{l_r\dot{\theta}}{v_x}\Big) \end{aligned}$$

These equations can be reduced to the following force system, dependent on yaw rate, vehicle side slip, and steering angle ($\dot{\theta}$, β, and δ respectively):

$$F_y = K_{F_y\dot{\theta}}\dot{\theta} + K_{F_y\beta}\beta + K_{F_y\delta}\delta \tag{3.3}$$

and

$$M_z = K_{M_z\dot{\theta}}\dot{\theta} + K_{M_z\beta}\beta + K_{M_z\delta}\delta \tag{3.4}$$

The K terms in equations 3.3 & 3.4 represent the following force system coefficients:

$$\begin{aligned} K_{F_y\dot{\theta}} &= \frac{\partial F_y}{\partial\dot{\theta}} = \frac{-C_f l_f\cos(\delta) + C_r l_r}{v_x} \\ K_{F_y\beta} &= \frac{\partial F_y}{\partial\beta} = -C_f\cos(\delta) - C_r \\ K_{F_y\delta} &= \frac{\partial F_y}{\partial\delta} = C_f\sin(\delta)\Big(\beta - \delta + \frac{l_f\dot{\theta} + v_y}{v_x}\Big) + C_f\cos(\delta) \\ K_{M_z\dot{\theta}} &= \frac{\partial M_z}{\partial\dot{\theta}} = \frac{-C_f l_f^2 - C_r l_r^2}{v_x} \\ K_{M_z\beta} &= \frac{\partial M_z}{\partial\beta} = -C_f l_f + C_r l_r \\ K_{M_z\delta} &= \frac{\partial M_z}{\partial\delta} = C_f l_f \end{aligned}$$

The force system defined in equations 3.3 & 3.4 can then be used to calculate lateral and yaw acceleration from the current lateral velocity and yaw rate given total vehicle mass m and rotational inertia about the vertical axis through the vehicle centre of mass I_z:

$$\begin{aligned} \begin{bmatrix}\dot{v}_y \\ \ddot{\theta}\end{bmatrix} &= \begin{bmatrix} \frac{K_{F_y\beta}}{mv_x} & \frac{K_{M_z\beta}}{I_z v_x} \\ \frac{K_{F_y\dot{\theta}}}{m} - v_x & \frac{K_{M_z\dot{\theta}}}{I_z}\end{bmatrix}\begin{bmatrix} v_y \\ \dot{\theta}\end{bmatrix} + \begin{bmatrix}\frac{K_{F_y\delta}}{m} \\ \frac{K_{M_z\delta}}{I_z}\end{bmatrix}\delta \\ &= \begin{bmatrix} \frac{-C_f\cos(\delta) - C_r}{mv_x} & \frac{-C_f l_f + C_r l_r}{I_z v_x} \\ \frac{-C_f l_f\cos(\delta) + C_r l_r}{mv_x} - v_x & \frac{-C_f l_f^2 - C_r l_r^2}{I_z v_x}\end{bmatrix}\begin{bmatrix} v_y \\ \dot{\theta}\end{bmatrix} + \begin{bmatrix}\frac{C_f\sin(\delta)}{m}\Big(\beta - \delta + \frac{l_f\dot{\theta} + v_y}{v_x}\Big) + \frac{C_f\cos(\delta)}{m} \\ \frac{C_f l_f}{I_z}\end{bmatrix}\delta \end{aligned}$$

Comparing the kinematic model to the dynamic model with the same inputs, we can see that the kinematic model follows a perfect circle as if it were on rails, while the dynamic model slips outward like a real vehicle would under high speed or low traction conditions.

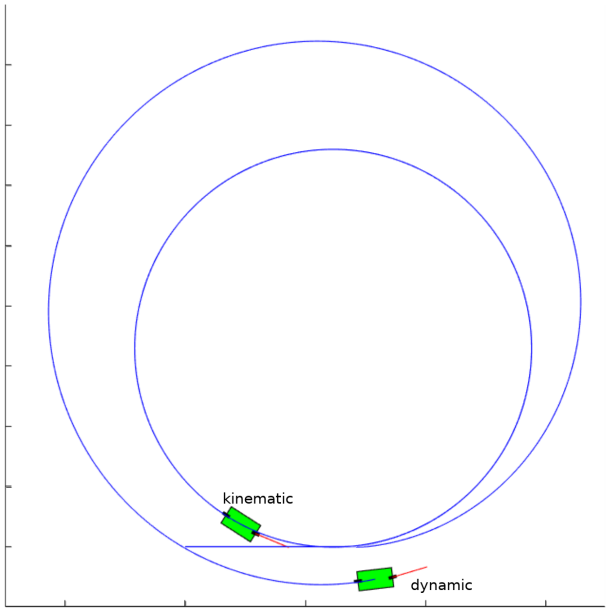


Figure 3.16: Comparison of kinematic and dynamic models with same inputs

The kinematic and dynamic motion models were then calibrated on the training datasets collected earlier. In order to compare and evaluate the motion models only, the models were fed with ground truth steering wheel angle and forward speed.

Figure 3.17 below shows the mean position error over time for the kinematic (green) and dynamic (blue) models. The solid line represents the mean error, while the shaded region indicates ± 1 standard deviation away from the mean error. The models were evaluated separately on the low speed (left) and high speed (right) datasets.

Figure 3.18 shows the same mean position errors but graphed against distance traversed by the ground truth trajectory. Since the trajectories collected are of differing time duration and traversed distance, the low speed results only considers the 51/107 trajectories longer than 40 seconds, or the 64/107 trajectories longer than 20 m. The high speed results considers the 53/74 trajectories longer than 40 seconds, or the 71/74 trajectories longer than 40 m.

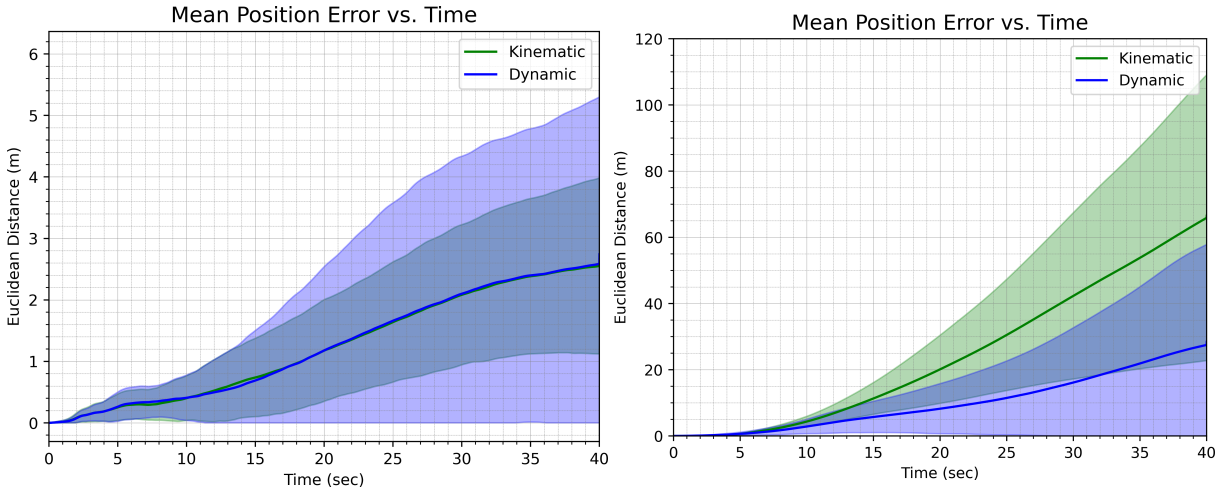


Figure 3.17: Position error over time for low speed (left) and high speed (right) datasets

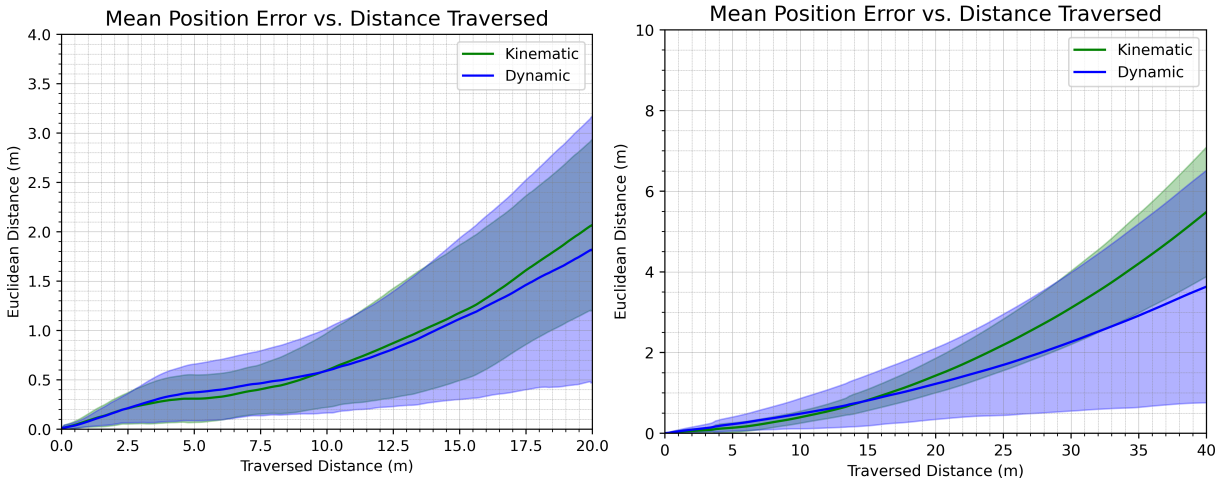


Figure 3.18: Position error over distance for low speed (left) and high speed (right) datasets

On the low speed dataset (Figures 3.17 & 3.18, left), the kinematic and dynamic models show very similar mean position error. However the kinematic model appears to be more consistent, showing lower standard deviation in error (indicated by the smaller green shaded region compared to the large blue shaded region).

Meanwhile on the high speed data (Figure 3.17 & 3.18, right) the kinematic model shows increasingly higher mean error past 7.5 sec or 17 m traversed. This is likely because the kinematic model doesn't account for lateral slip which becomes non-negligible at higher speeds. Under low speed conditions with negligible slip, the kinematic and dynamic models perform similarly well. Considering the accuracy of both models on the high speed data however, both models perform quite poorly. Tuning the models for across a wide range of speeds was very difficult, so work was focused on achieving the best accuracy under low speed conditions, which would be where the vehicle would operate while performing precision navigation to dock with a payload.

The trends seen in position error over time and distance also hold true for heading error; comparable performance at low speed, better performance from the dynamic model at high speed, and overall lower standard deviation in the kinematic model

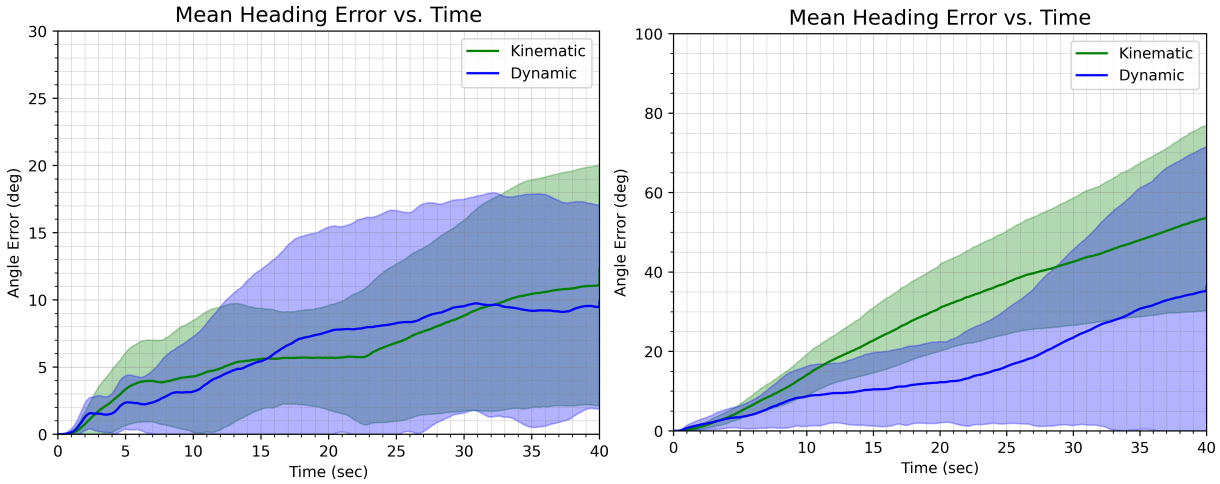


Figure 3.19: Heading error over time for low speed (left) and high speed (right) datasets

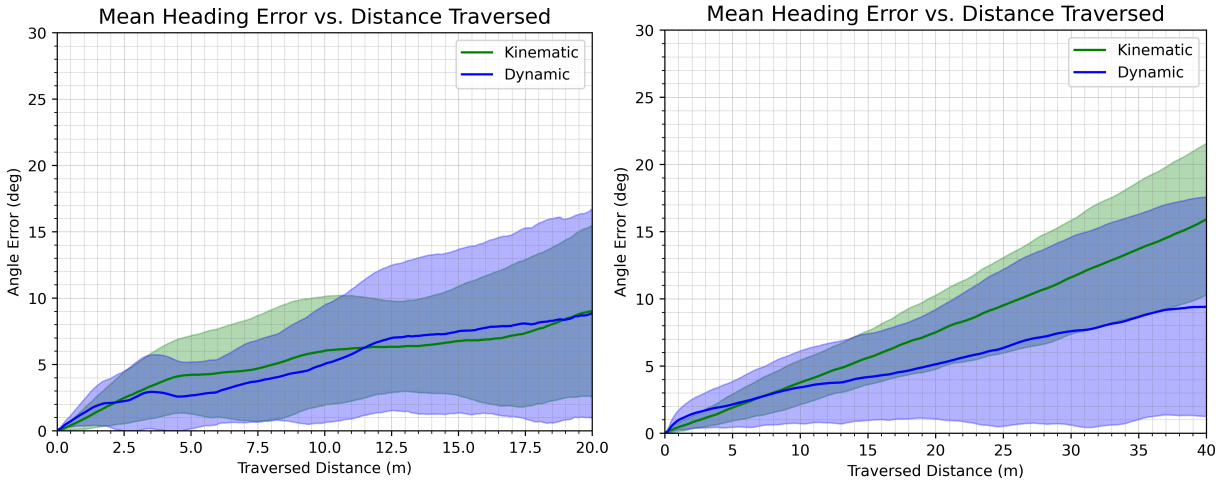


Figure 3.20: Heading error over distance for low speed (left) and high speed (right) datasets

3.4 Actuator Models

While the aforementioned motion models are sufficient to model vehicle motion given vehicle speed and steering, these variables are state variables of the system rather than input variables. The input variables available for our system are target speed command, target throttle position, and target steering wheel angle. In order to predict vehicle speed and steering to feed into our motion models, we need to also model these actuators.

3.4.1 Steering model

The steering wheel is actuated using a motor driven by a controller with a builtin PID position control loop. A first order linear model was used to predict the steering wheel turning rate given a steering angle command, which is then integrated to give estimated steering wheel angle. The model was created with the following assumptions:

- There is a small constant delay between a new steering angle command being issued and the steering wheel changing velocity.
- The steering wheel turning rate is proportional to the difference between the current steering angle and the target steering angle, capped to a fixed maximum rate.
- The steering wheel turns faster when returning to centre than when turning away from centre.
- The steering wheel turns slower when the vehicle is moving slowly.

The steering model used can be expressed as follows:

Algorithm 1: Steering Model

for i = *1 to N* **do**
  steer_spd = (cmd[i-delay] - steer_angle[i-1]) $\times$steer_spd_constant;
  steer_spd = clip(steer_spd, MIN_SPD, MAX_SPD);
  if *steering to centre* **then**
    steer_spd = steer_spd $\times$K$_A$; // $K_A \geq 1$
  end
  K_B = clip($K_C \times \frac{\text{vehicle\_speed}}{K_D}$, MIN_DEBUFF, 1);
  steer_spd = steer_spd $\times$K$_B$; // MIN_DEBUFF$\geq K_B \geq 1$
  estimated_steer_angle[i] = estimated_steer_angle[i-1] + steer_spd $\times t$;
end

Figure 3.21 shows a plot of the steering model being applied. The mean absolute error for the predicted steering angle over this run is 2.76°.

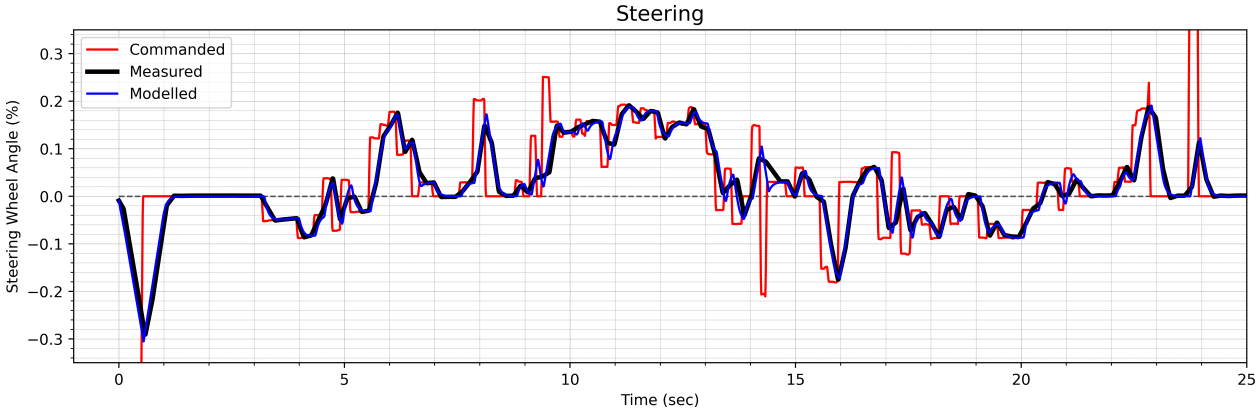


Figure 3.21: Steering Model

3.4.2 Forward speed model

For the steering model, we modelled the physical system lumped together with the position controller which took steering angle commands as inputs. For the speed model however, we have the option of modelling the speed using speed commands or using throttle position commands.

If we apply the same actuator model used by Seegmiller [5] shown in Equations 3.1 & 3.2 and tune it to the Yamaha vehicle under our usecase, we get the following result shown below in Figure 3.22:

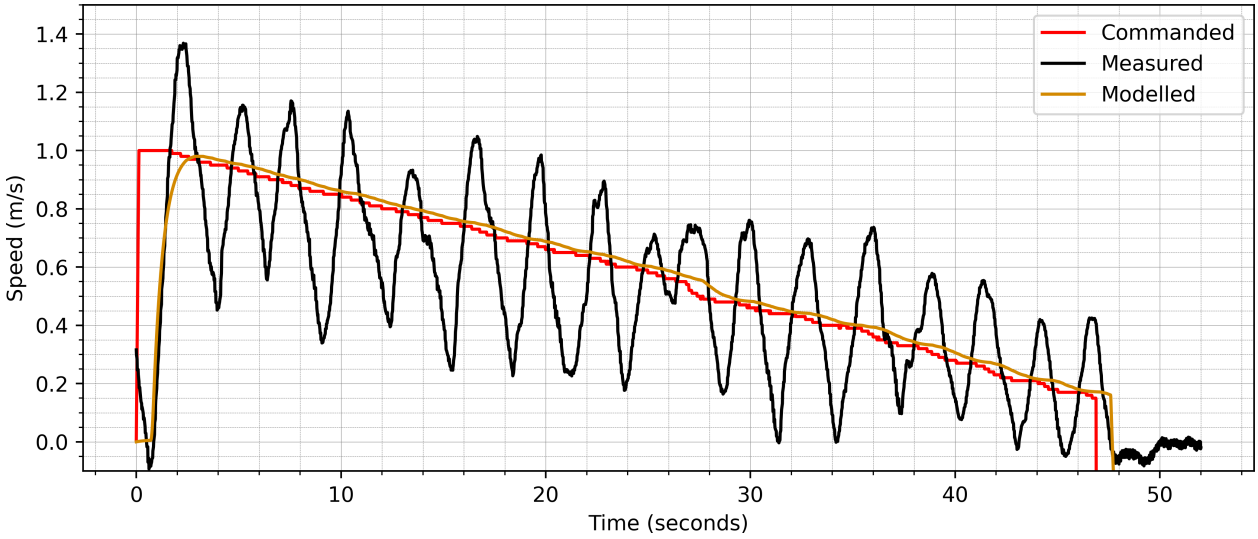


Figure 3.22: First order speed model applied to the Yamaha vehicle at low speed

The result we get with this model is significantly worse than the result obtained when the model was applied to the LandTamer platform (Figure 3.4), due to the large oscillations of the vehicle speed about the setpoint. While all the vehicles used in past work with this speed model were driven by electric motors, our Yamaha UTV is driven by an internal combustion engine with a significantly more complex transmission system, and struggles to maintain a constant low speed (see also Figure 2.24).

Even at higher vehicle speeds, if the speed controller is unable to accurately track the speed setpoint, it is very difficult to predict vehicle speed from the speed command alone, especially with a simple first order linear model. An approach with a more complex model showed some promise however; Figure 3.23 shows results of preliminary work on predicting vehicle speed with only speed commands using a machine learning approach. This approach uses a recurrent neural network (RNN) and 40 minutes of driving data with varying speed commands. The RNN predicts vehicle speed from speed command, in steps of 50 ms. This work was done prior to the finalisation of the vehicle speed controller, so the vehicle is not tracking the commanded speed as accurately as it does in the final version.

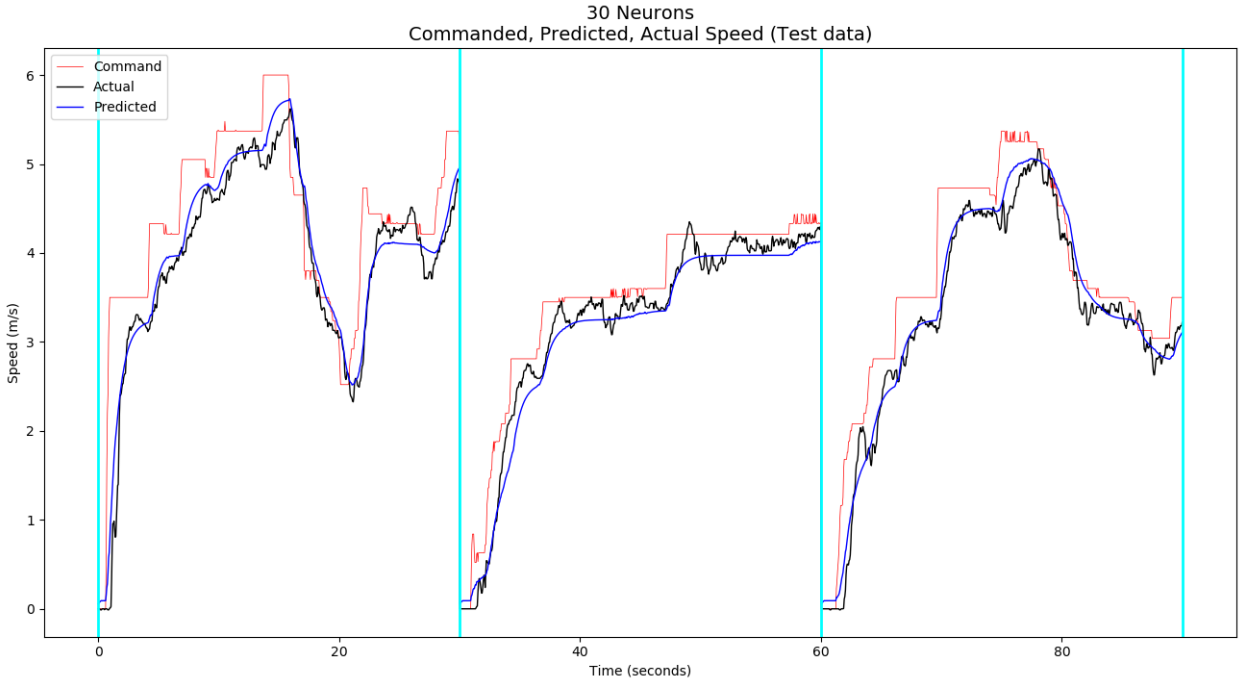


Figure 3.23: Vehicle speed prediction using a recurrent neural network

While this approach showed promise, it would need to be retrained whenever a change was made to the vehicle speed controller since the model lumps the controller together with the physical system. It also required a significant amount of data to train each time, and although there are approaches to alleviate this issue such as training from previous models, or freezing network weights, it still required a lot of work for just a speed model.

Returning to the first order model, the initial attempts to model speed (Figure 3.22) showed that the model output is simply tracking the commanded speed. With the speed command alone, particularly at low speed, there is simply not enough information to accurately predict the vehicle speed. Another approach we could take, building upon the first order model, is to use throttle command as the model input, taken from the output of the vehicle speed controller.

With this idea in mind, a new speed model was made which uses throttle command to predict forward acceleration of the vehicle, which can then be integrated to obtain forward velocity. The rationale behind this was that the throttle pedal in a passenger internal combustion car doesn't command speed, but rather commands either power or torque. This can be seen when holding the throttle pedal at a fixed position while driving along a road with varying slope; the vehicle speed doesn't stay constant while the slope (and consequently load on the engine) changes, instead it increases up to some point where the engine output balances with the load from gravity, friction, and drag. The model is still a first order linear model, and assumes the vehicle is travelling along level ground at low speed, ignoring loading due to gravity and air resistance. A constant friction term and linear drag term are also included in the model to account for friction and drag in the powertrain, engine braking, rolling resistance, etc.

$$v[i] = v[i-1] + \Big(\text{throttle\_cmd}[i-\text{delay}] \cdot K_{const} - \big(K_{friction} + K_{drag} \cdot \big|v[i-1]\big|\big)\Big) \cdot \Delta t \qquad (3.5)$$

The result of this model applied to the same run shown previously in Figure 3.22 is plotted below in Figure 3.24:

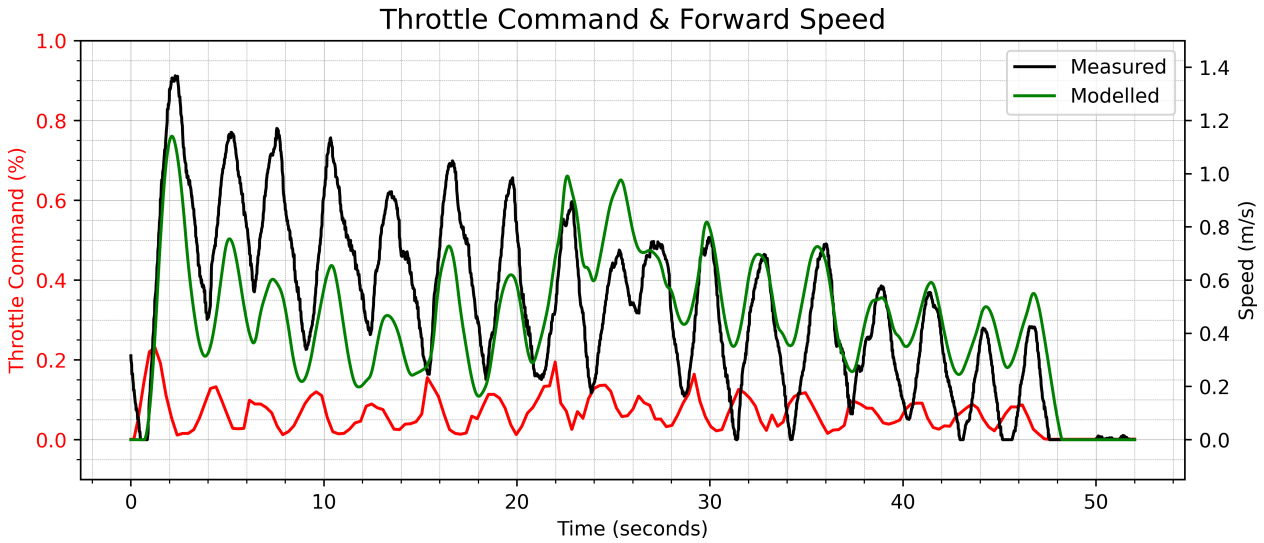


Figure 3.24: First order acceleration model applied to the Yamaha vehicle at low speed

This model significantly improves upon the previous model, with the predicted speed varying as the measured speed oscillates about the setpoint. This approach also has the benefit of not needing to be recalibrated every time the vehicle speed controller is modified. However it still doesn't accurately track the measured vehicle speed, underestimating speed in the first portion of the plot and overestimating speed towards the end. With this model formulation, many assumptions and approximations were made in order to minimise the number of parameters and keep the model easy to calibrate. Approximations include linear throttle response, linear drag force, and a fixed constant friction force. In addition to this, the entire drivetrain of the vehicle was lumped into one fixed constant term. While this simplification may be appropriate for an electric powertrain with fixed or no gearing, our Yamaha UTV has a complex automatic transmission system which is constantly adjusting depending on the current conditions. In order to obtain a more accurate model, more detail would need to be put into how the drivetrain actually functions.

Drivetrain Analysis

This vehicle features Yamaha's Ultramatic® transmission. The key components of this system are centrifugal clutch, variable-diameter pulley continuously variable transmission (VDP CVT), and a one way sprag clutch. With this system, the vehicle automatically engages the engine output to the transmission with the centrifugal clutch as throttle is applied, and dynamically adjusts the effective gear ratio using the CVT depending on engine output and wheel load. Finally the additional sprag clutch enables automatic engine braking independent of the centrifugal clutch.

Figure 3.25 shows the first stage of the transmission, where the output from the engine (the engine crankshaft) connects to the centrifugal clutch carrier assembly (5). As the engine speed increases, the weights in this clutch are forced radially outwards, engaging with a clutch housing that it is housed within (1).

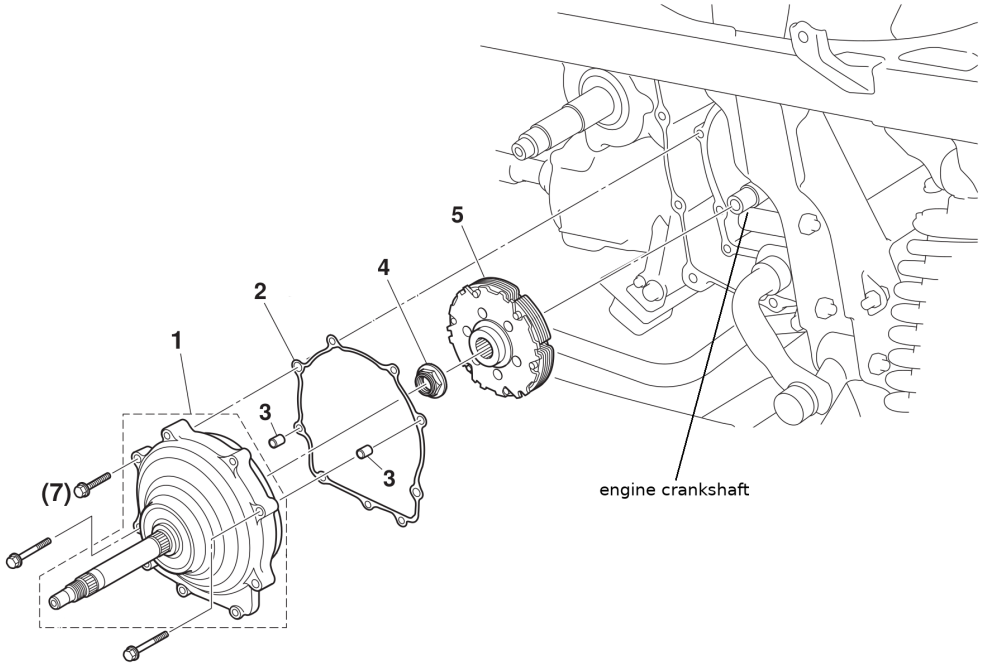


Figure 3.25: Centrifugal clutch assembly [10, p. 5-60]

Figure 3.26 shows the internals of the clutch housing assembly from Figure 3.25 (1). In Figure 3.26, the clutch housing (9) is the component which engages with the centrifugal clutch pads as they expand outwards. This is the primary interface through which torque is transferred between the engine and the rest of the transmission. The secondary interface is through the one-way clutch bearing (7). When the clutch housing is rotating slower than the engine crankshaft, this bearing freewheels and can effectively be ignored. When the clutch housing tries to rotate faster than the engine crankshaft however, the sprags in this one-way bearing lock, and transfer torque from the clutch housing through this bearing to the engine crankshaft, resulting in engine braking. These two mechanisms operate independently of each other.

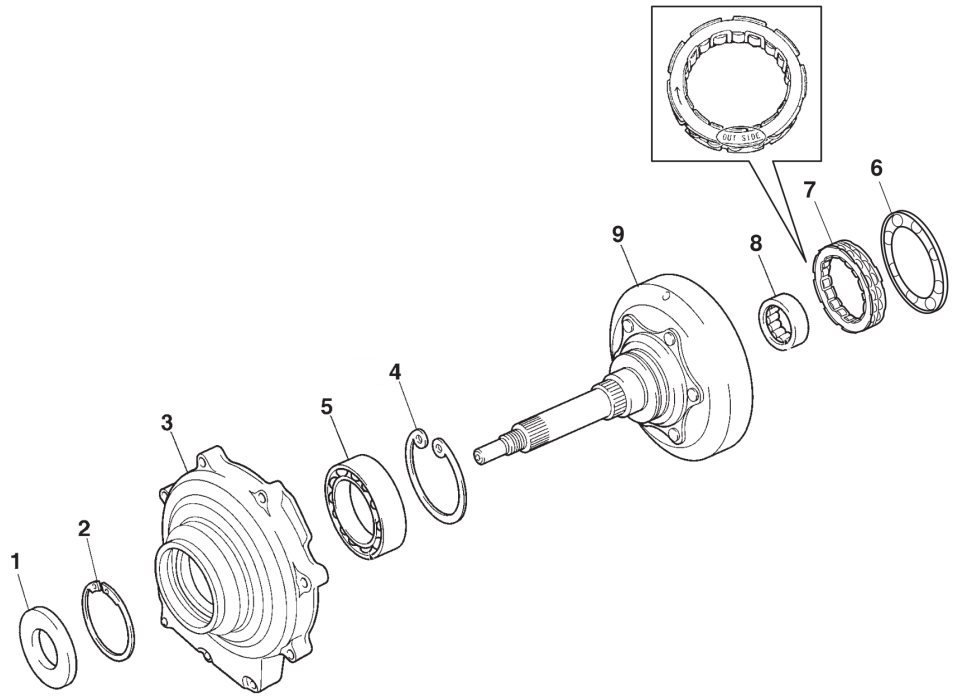


Figure 3.26: Centrifugal clutch housing internal assembly [10, p. 5-61]

The clutch housing (9) has a shaft with splines, and protrudes from the bearing housing (3) to connect to the CVT primary pulley assembly. This is the first of the two pulleys which make up the belt drive CVT system.

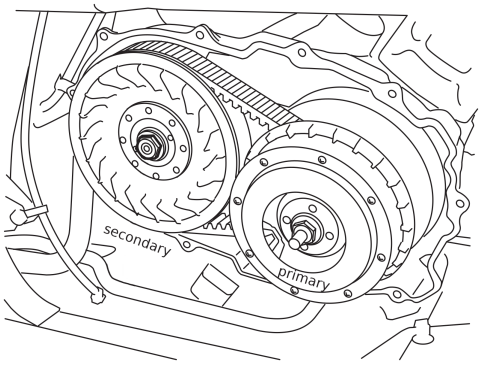


Figure 3.27: CVT assembly shown mounted on the vehicle [10, p. 5-58]

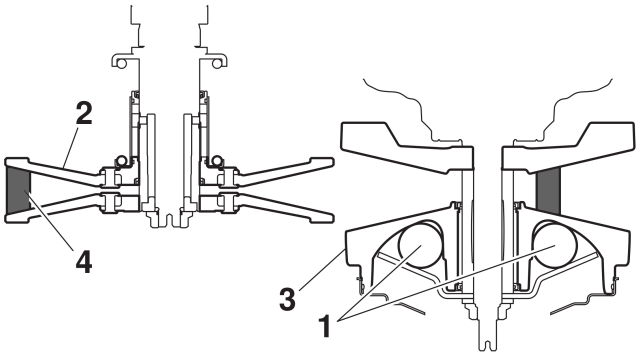


Figure 3.28: Top-down cross-sectional view of CVT assembly [10, p. 5-58]

Figure 3.27 shows the CVT assembly when mounted on the vehicle connected by a v-belt.

In Figure 3.28, the splined shaft of the centrifugal clutch housing is inserted through the two sheaves (3) which make up the primary pulley. The secondary pulley also has two sheaves (2) and

is connected to the first pulley by a v-belt (4). Both pairs have sheaves have adjustable separation distance, which depend on the vehicle's wheel load, engine torque, and engine speed.

At idle, a spring loaded mechanism in the secondary pulley forces the secondary sheaves together. The deconstructed mechanism can be seen in Figure 3.29. The spring (3) forces the sliding sheave (6) closer to the fixed sheave (8). This scenario is depicted in Figure 3.28, where the secondary pulley sheaves are forced together with minial separation, forcing the belt (4) outwards and increasing the pulley's effective diameter. This forces the belt at the primary pulley to move closer to the centre, forcing the primary pulley sheaves apart. In this configuration, the effective gear ratio is at its lowest, similar to starting a manual transmission in its lowest gear.

As throttle is applied and engine speed increases, the weights in the primary sheave assembly (Figure 3.28 component 1 and Figure 3.30 component 4) are forced radially outwards, sliding along slanted ramps. These weights push the the primary pulley sheaves together, increasing the effective diameter of the pulley. This force counterracts the spring force at the secondary pulley, forcing the secondary sheaves apart and reducing the secondary pulley's effective diameter. Through this mechanism, the effective gear ratio through the CVT is adjusted based on engine speed and torque, and wheel load.

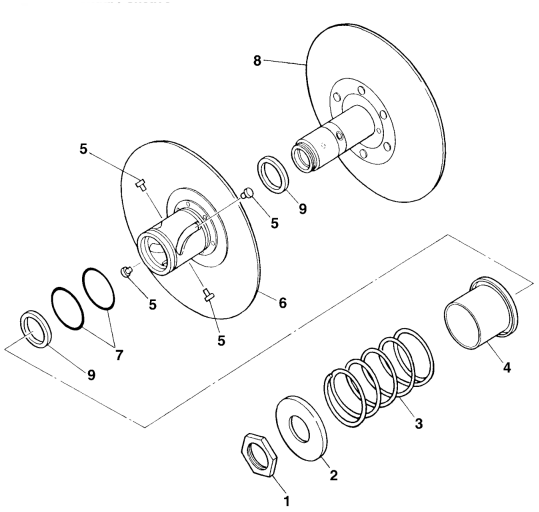


Figure 3.29: Secondary pulley assembly [10, p. 5-52]

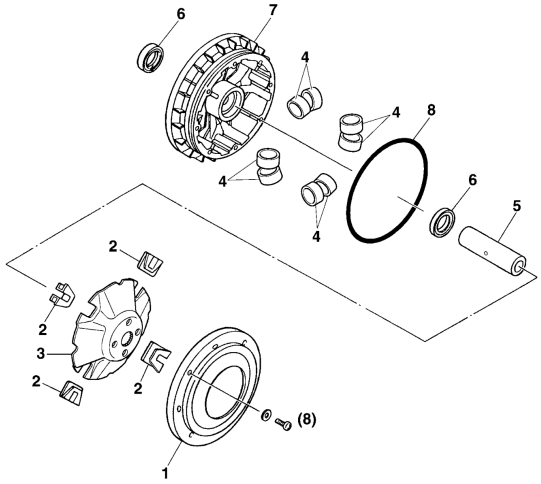


Figure 3.30: Primary pulley assembly [10, p. 5-51]

With this insight, the following model is proposed to predict vehicle speed given throttle commands:

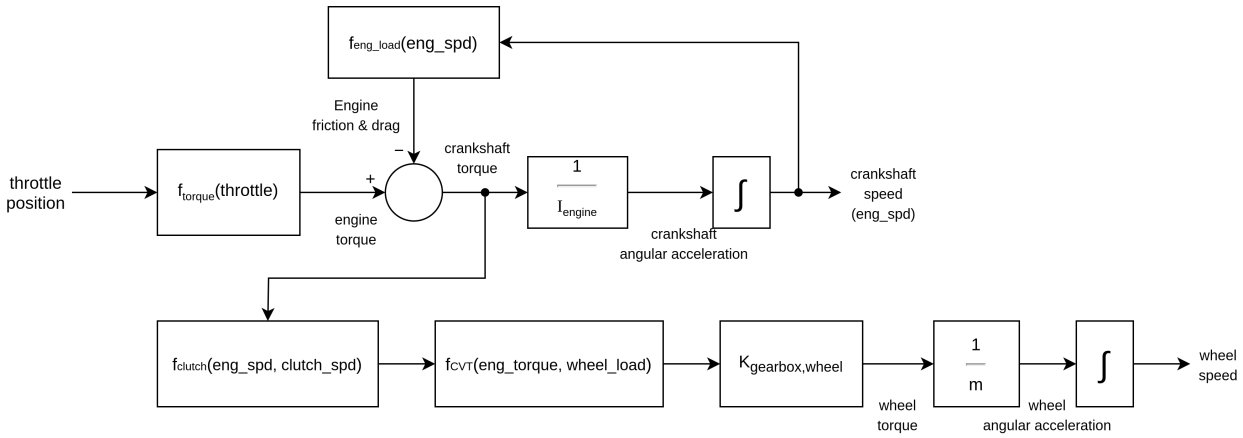


Figure 3.31: Speed model incorporating drivetrain

The model can be considered in two sections; the top row predicts crankshaft torque and speed using throttle position, while the bottom row predicts drive wheel speed from the previously predicted crankshaft torque. Our Yamaha vehicle has sensors measuring throttle position and wheel speed, and we can tune and validate using data from these two sensors with some scale ambiguity in the intermediate terms. If the vehicle also had an interface for logging engine RPM, then it would provide another constraint we could use to tune the model.

In the top section, throttle position percentage is mapped to generated torque with a logarithmic function. In reality, this function is dependent on additional parameters including current engine speed, temperatures, and is not purely logarithmic. However this approximation is an improvement over a linear mapping, and only requires three parameters to define.

$$f_{\text{engine torque}}(\text{throttle}) = K_{scale}\left(\log\left((\text{throttle}) + K_{x\_offset}\right) + K_{y\_offset}\right) \tag{3.6}$$

Where:

$$\begin{aligned} 10^4 < K_{scale} < 10^5 \\ 0 < K_{x\_offset} < 1 \\ \log(K_{x\_offset} + 0.1) < K_{y\_offset} < -\log(K_{x\_offset}) \end{aligned}$$

Figure 3.32 shows a sample of potential throttle curves.

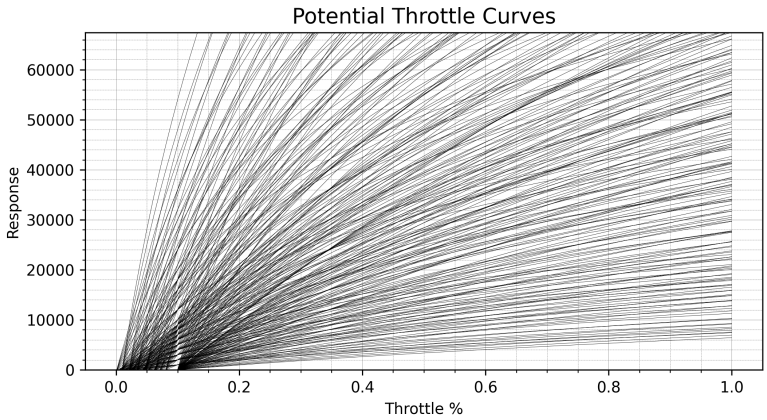


Figure 3.32: A sample of potential throttle curves

Engine load and wheel load are parameterised by a fixed friction term and a speed dependent drag term:

$$f_{\text{eng\_load}}(\text{eng\_spd}) = K_{\text{eng\_friction}} + K_{\text{eng\_drag}}|\text{eng\_spd}| \tag{3.7}$$

$$f_{\text{wheel\_load}}(\text{wheel\_spd}) = K_{\text{wheel\_friction}} + K_{\text{wheel\_drag}}|\text{wheel\_spd}| \tag{3.8}$$

The transfer of torque through the centrifugal clutch is dependent on the crankshaft speed. The proportion of torque transferred is assumed linearly proportional to the crankshaft speed up to the clutch lock speed- see Figure 3.33. The clutch_lock_speed is the only parameter that defines this stage.

Independently of the clutch, if the clutch housing speed is ever faster than the clutch carrier, the one-way bearing engages and 100% torque is transferred through this stage from the wheels to the engine.

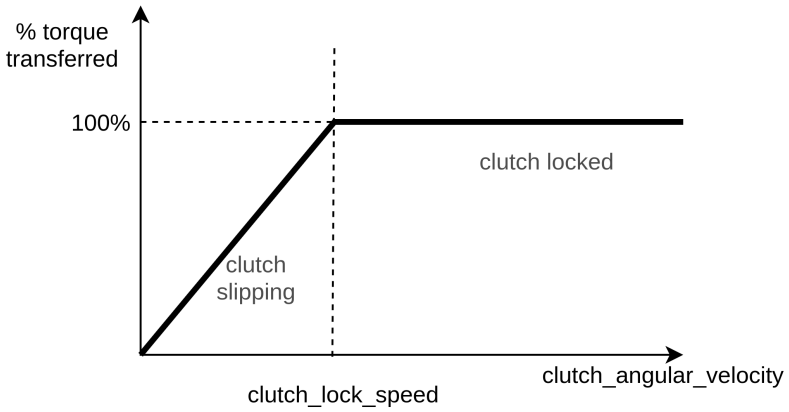


Figure 3.33: Centrifugal clutch torque transfer model

```
# Calculate torque transfer through centrifugal clutch
clutch_transfer = 1. # default to no clutch slip
# If the clutch carrier speed (=engine_speed) is below clutch slip speed
```

```
if eng_speed[i,1]<CLUTCH_LOCK_SPD:
  clutch_transfer = (CLUTCH_LOCK_SPD-eng_speed[i,1])/CLUTCH_LOCK_SPD #proportional
# if the relative speed is negative, one-way bearing transfers all torque
clutch_housing_spd = wheel_speed[i-1,1] * K_fixedtrans * K_cvt
clutch_speed_diff = eng_speed[i,1] - clutch_housing_spd
if clutch_speed_diff<0:
  clutch_transfer = 1.
```

The CVT ratio is determined by the ratio between wheel load propagated backwards through the gearbox, and the engine output torque propagated through the clutch and CVT to the secondary sheaves. This ratio is assumed to vary over time in fixed steps over a fixed range.

```
# Calculate CVT ratio
CVT_primary_torq = eng_torq * clutch_transfer * K_cvt * K_cvt_const
wheel_load = WHEEL_FRICT[0] + WHEEL_FRICT[1]*wheel_speed[i-1,1]
CVT_secondary_torq = wheel_load * (1./K_fixedtrans)
if CVT_primary_torq>CVT_secondary_torq:
  K_cvt -= CVT_step_sz
elif CVT_primary_torq<CVT_secondary_torq:
  K_cvt += CVT_step_sz
K_cvt = np.clip(K_cvt, K_cvt_min, K_cvt_max)
```

Finally the $K_{\text{gearbox,wheel}}$ is a lumped constant incorporating the gearbox where the driver can select between a high/low/reverse gear, and any fixed multiplication and scaling converting driveshaft angular velocity into forward speed.

The model was tuned on the low speed training dataset. Figure 3.34 shows a plot of the two first order models applied to a run from the test dataset.

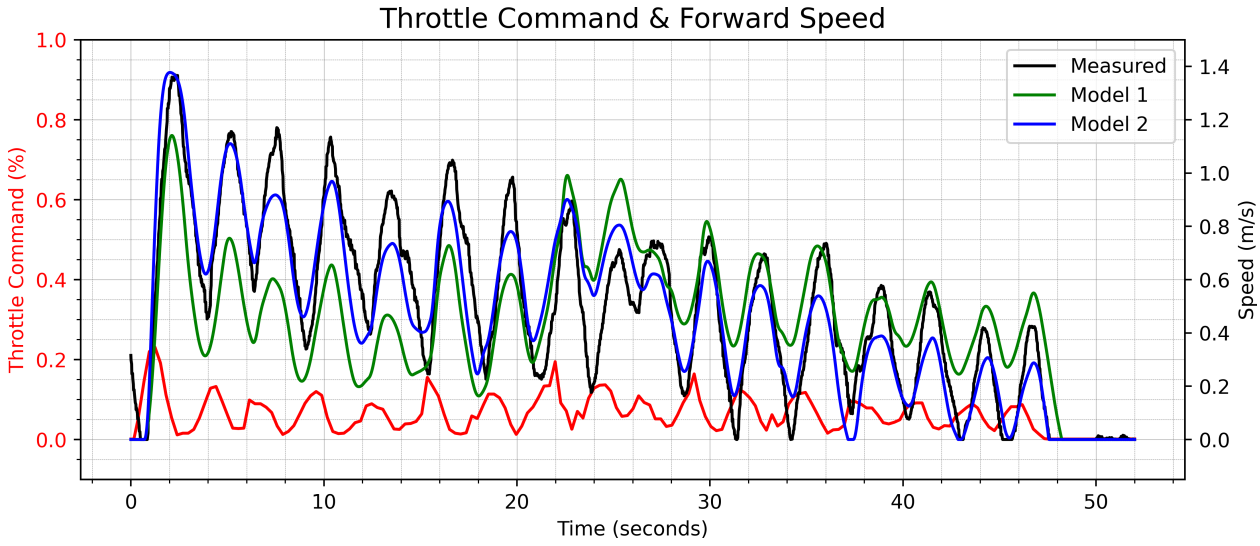


Figure 3.34: Speed model incorporating drivetrain (Model 2) compared to simple first order model (Model 1)

Across the low speed training, validation, and test datasets, we see a reduction in mean MSE of 31%, 32%, and 27% respectively.

3.5 Learned models

In all the models encountered so far, many assumptions had to be made in order to have a tractable model. An alternative method is to use machine learning techniques to train a model to predict system response to input commands, using data collected during field testing.

3.5.1 Pure Learning-Based Model

In section 3.2 we discussed work done by Bode on a data-driven approach for learning a forward predictive model [6]. Bode's method predicts along-track, cross-track, and yaw velocities for a skid-steer vehicle using current velocities, input commands, and vehicle roll & pitch. The inputs are fed through a fully connected feed forward neural network to predict the next time step, and multiple time steps at a time are predicted by chaining these networks together and assuming roll & pitch do not change from their initial condition. We can apply a similar approach modified to suit our Yamaha vehicle under low speed driving.

Training was done on the low speed training dataset. To evaluate, the model was run on the low speed test dataset to generate predicted trajectories. The euclidean distance between predicted position and GPS measured position was calculated for each corresponding time step and ground truth path distance step. Figure 3.35 shows the mean and standard deviation for position error over time (left) and over distance (right). The trajectories were divided into 5 second segments when evaluating mean error over time, while the error vs distance plots average the 52 out of 123 trajectory segments that are longer than 3.5 m.

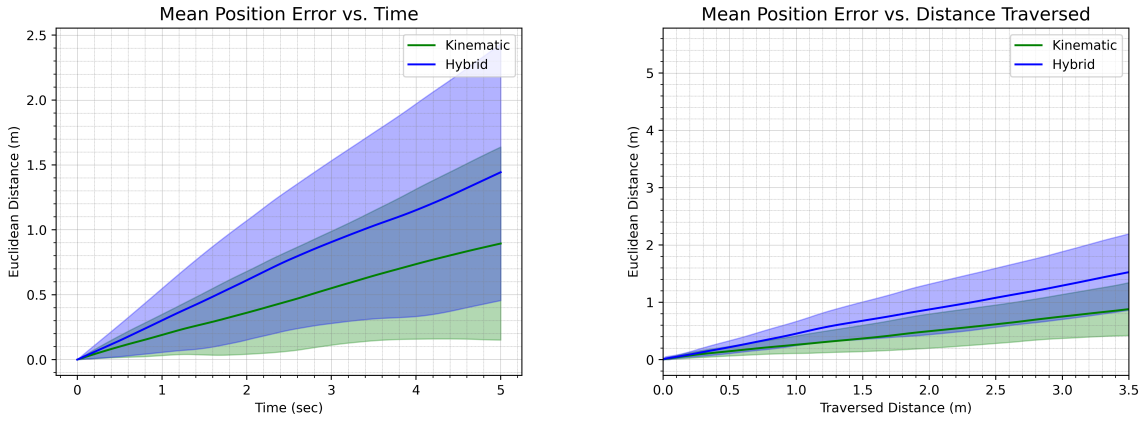


Figure 3.35: Mean position error of the feedforward neural network model and kinematic model over time (left) and distance (right) on the low speed test dataset

Unfortunately the neural network (blue) performs even worse than the kinematic model (green). The kinematic model here uses our actuator models rather than ground truth vehicle speed and

steering, so it is predicting vehicle motion given steering and throttle commands. The neural network is using speed & steering commands and vehicle roll & pitch to predict velocities, which are integrated and coordinate transformed to get predicted positions relative to an inertial frame. As shown in Figure 3.24, the speed controller has difficulty tracking the speed command at low speed, while the neural network is using only speed commands as a basis for predicting vehicle speed. In addition to this, while Bode had 7 hours of data to work with [6], we've got only 44 minutes of data to train on, with which the model needs to learn the behaviour of the steering and powertrain, as well as how a vehicle behaves, from scratch with no prior knowledge.

3.5.2 Hybrid Models

To improve upon the results of the previous subsection, a few changes are made to the approach: First, instead of using speed commands as an input, we'll use throttle commands which offer more information. Secondly, we're working with time series data on a system which has delays between its inputs and state variables, so an input may not affect the state until some number of time steps into the future. We need our model to take into account previous inputs, either through retaining them in a hidden state with recurrent neural network architectures, or by directly providing them in the input layer. Finally, we need to introduce prior knowledge of vehicle behaviour into the model so isn't learning how a vehicle behaves from scratch.

We decided to start with a hybrid approach combining the kinematic bicycle model with a feedforward neural network, similar to approach (c) in Figure 3.8. The kinematic bicycle was chosen as the physical basis model as it offered similar accuracy and higher consistency compared to the dynamic bicycle model at both low speed, and at high speed over short prediction horizons. This can be seen in Figures 3.17 & 3.18 of the previous section, which show the two models having similar mean error, but the kinematic model having lower standard deviation in error. In addition to this, the kinematic model is less computationally expensive, and produces fewer outputs which need to be fed into the neural network; since there is a no-slip assumption, lateral slip velocity from the kinematic model is always 0.

The input features to the neural network are steering angle percentage command ([-1,1]), throttle position percentage command ([0,1]), the predicted steering angle percentage ([-1,1]), predicted forward speed (m/s), and predicted yaw rate from the kinematic model (radians/second).

Since there are delays in the physical system between the inputs and output, for each prediction time step we also need to provide the model with inputs sampled from past time steps; for each feature we used the past two seconds of data sampled at 20 Hz. These 40×5 samples are concatenated and fed into the input layer.

For the hybrid architecture, we tried two different approaches for combining the kinematic model with the learned model. The first approach shown in Figure 3.36 has the neural network learn the vehicle frame longitudinal, lateral, and yaw velocities directly, and integrates these to get position. In the second approach shown in Figure 3.37, the neural network learns the residuals for the velocities predicted by the speed model and kinematic model. The summation of the velocity predictions and the residuals predictions are then integrated as with the first approach.

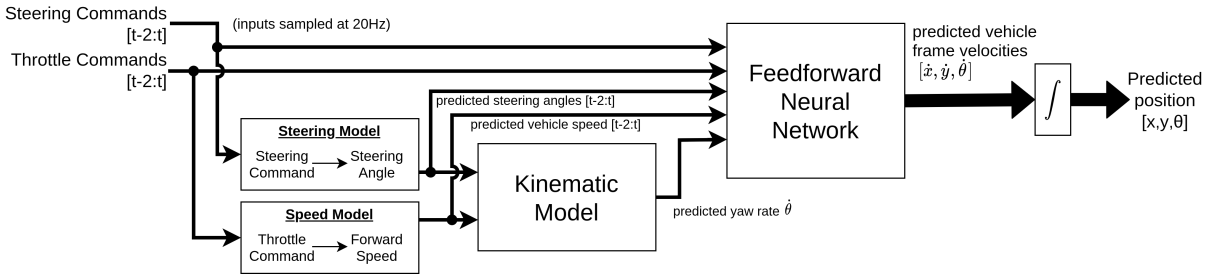


Figure 3.36: Hybrid feedforward model learning velocity directly.

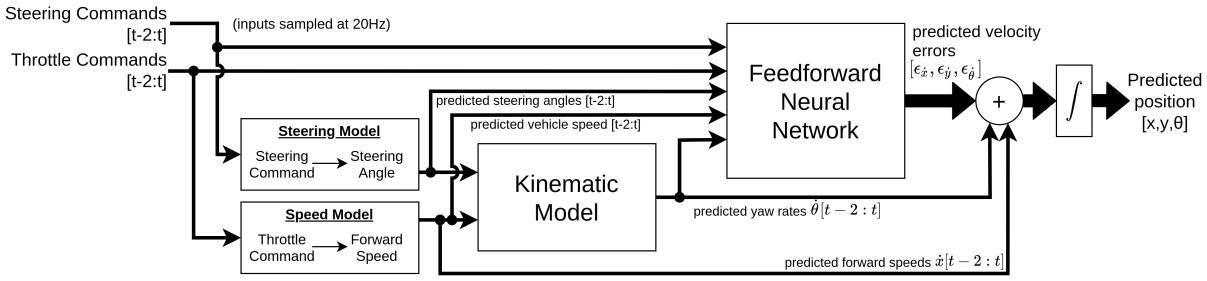


Figure 3.37: Hybrid feedforward model learning velocity residuals

The two hybrid model architectures are compared using identical hyperparameters on the low speed test dataset . Figure 3.38 shows the mean and standard deviation of the position error (left) and heading error (right) over time for the two models.

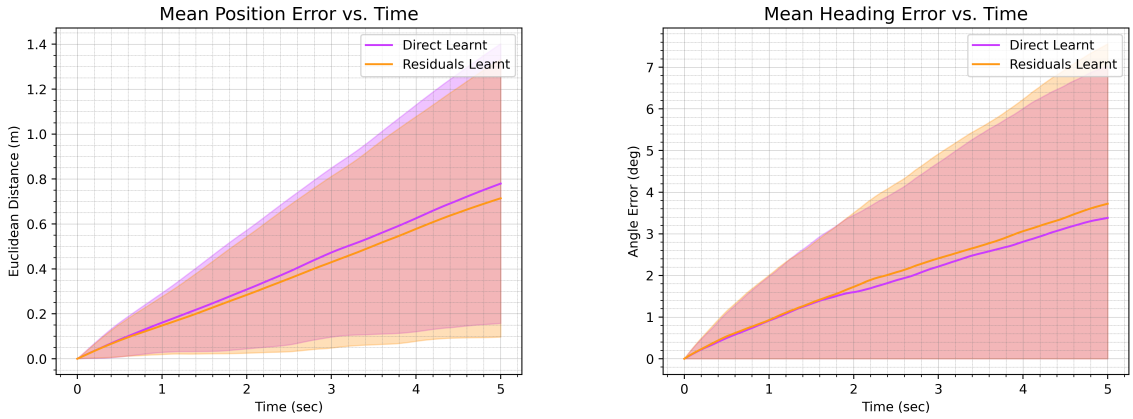


Figure 3.38: Comparison between direct learning and residuals learning hybrid model architectures for position error (left) & heading error (right) over time on the low speed test dataset

The above plots show that the two hybrid models are fairly comparable; the final mean error for the residuals learning model is only 6.5 cm less than for the direct learning model, while the final heading error for the direct learning model is 0.3° lower than for the residuals learning model.

We can also compare the errors over distance rather than time; again we compare only the 52/123 trajectories where distance travelled is further than 3.5 m:

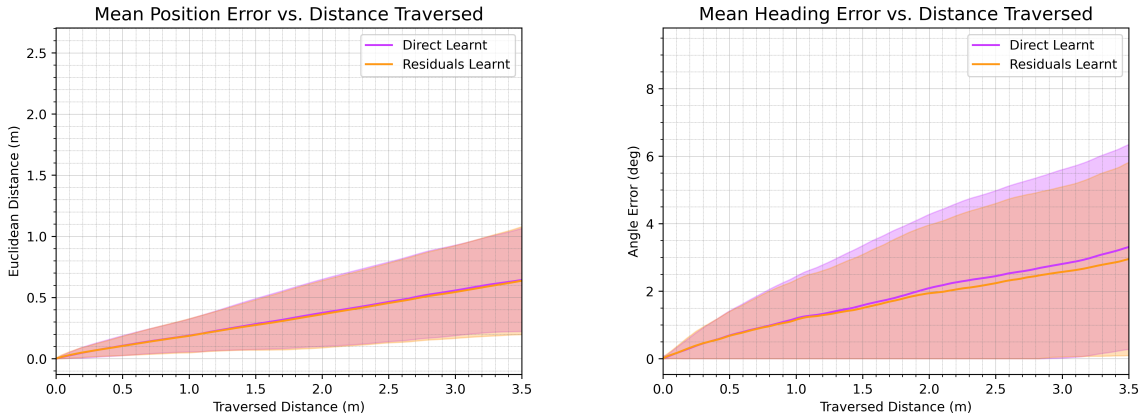


Figure 3.39: Comparison between direct learning and residuals learning hybrid model architectures for position error (left) & heading error (right) over distance on the low speed test dataset

In the error vs distance plots above, the position errors over time for the two models almost identical, but the final mean heading error for the residuals learning model is now 0.3° lower than for the direct learning model.

Overall the two appear to be fairly comparable performance wise. Cost-wise, learning residuals just takes an extra addition step. We decided to proceed with the residuals learning approach, simply because its mean position error was slightly lower in these tests.

The graphs in Figure 3.40 below compare mean position & heading errors over time & distance for the residuals learning hybrid model compared to the kinematic model.

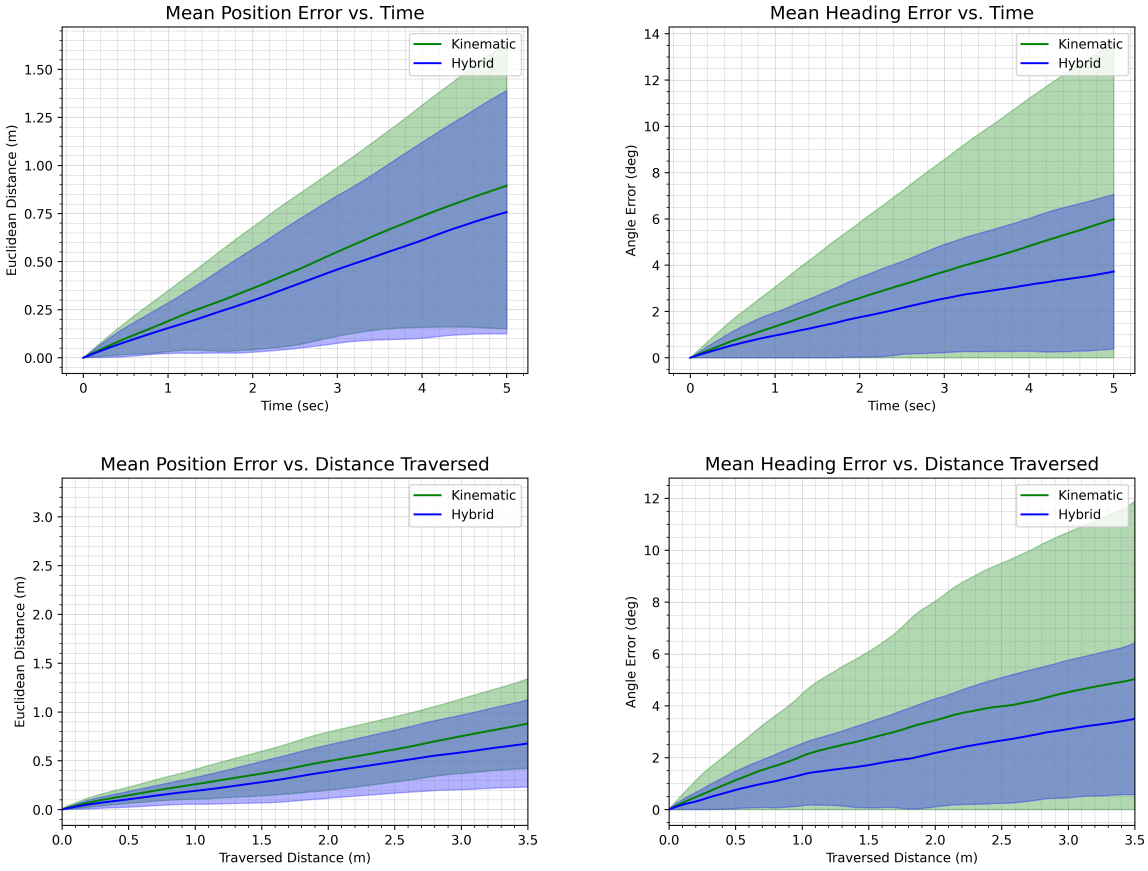


Figure 3.40: Comparison between the residuals learning feedforward hybrid model and the kinematic model for position & heading error over time & distance on the low speed test dataset

In all graphs, the hybrid model (blue) outperforms the kinematic model (green). The final mean position error is 14 cm (15%) less at 5 seconds or 20 cm (23%) less at 3.5 m. The final mean heading error is 2.3° (38%) less at 5 seconds or 1.5° (30%) less at 3.5 m.

Instead of using a feedforward neural network and feeding in multiple samples per feature at each time step, we could use a recurrent neural network architecture that learns a representation for the state and carries it forward through successive time steps. For the neural network architecture, we tried recurrent neural network (RNN) cells, long short-term memory (LSTM) cells, and gated recurrent unit (GRU) cells. RNNs similar to feedforward neural networks but successive predictions are dependent on the state of units from the previous prediction. LSTMs improve upon the RNNs by including a memory cell which helps circumvent the vanishing/exploding gradient issues which can arise in RNNs. This helps LSTMs perform better over long prediction horizons [22], which is important for modelling systems which can have long delays such as an internal combustion engine powertrain. The GRU is a newer recurrent architecture which also controls the flow of information between successive predictions like an LSTM but does so using a different, simpler method. The GRU was found to be comparable to the LSTM in accuracy, but is less complex and learns faster [23]. In certain cases however such as very long prediction horizons, the LSTM can still outperform the GRU [24].

Figures 3.41 and 3.42 outline the architecture of the hybrid models utilising recurrent neural networks. Similar to the feedforward neural network hybrid models previously shown, we use the same five input features. However this time we feed in only one sample per feature per time step.

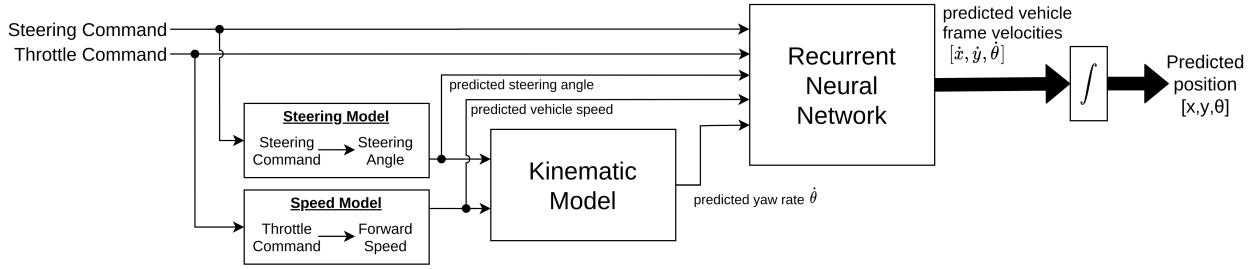


Figure 3.41: Hybrid recurrent model learning velocity directly.

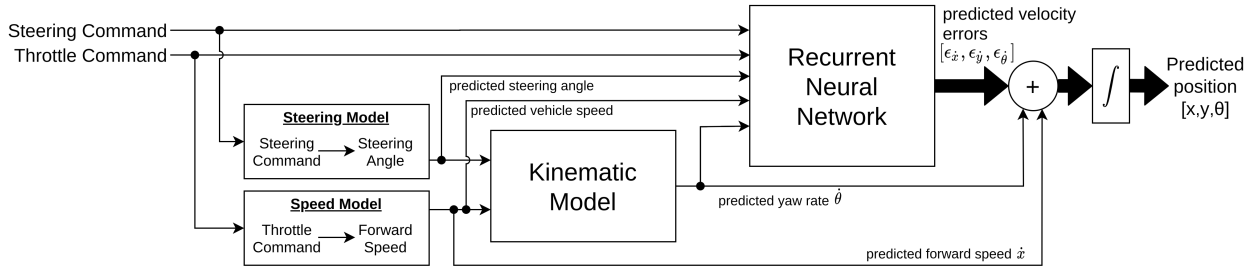


Figure 3.42: Hybrid recurrent model learning velocity residuals

Comparing mean error from hybrid models predicting velocity residuals using RNN, LSTM, or GRU cells:

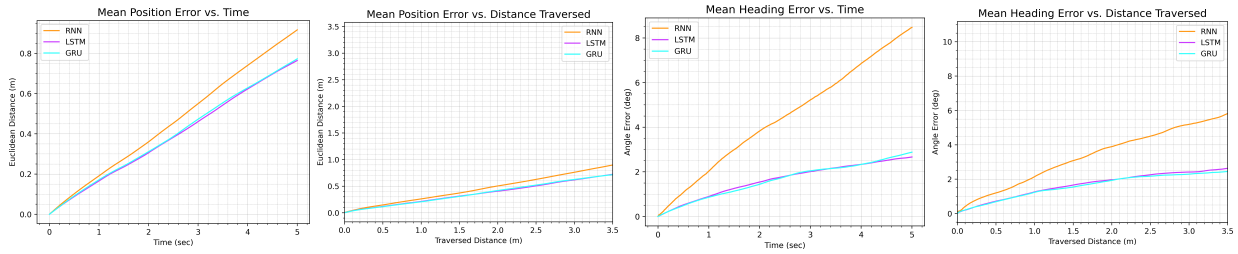


Figure 3.43: Mean position & heading error comparison between RNN, LSTM, and GRU cells in hybrid models predicting velocity residuals

As expected from theory, the LSTM (magenta) and GRU (cyan) based hybrid models are almost identical in accuracy. The RNN based model (orange) has higher mean error than the other two model types, likely due to the RNN cell's poorer performance in learning long-term dependencies. Using time step sizes of 10 ms, if the throttle has a delay of 0.5 seconds then a change in throttle will not affect speed until 50 time steps later, so the recurrent neural network has to retain information within its hidden state for that many timesteps.

In terms of model size, for the same number of layers and hidden units, the simple RNN had about $\frac{1}{4}$ the number of trainable parameters as the LSTM, while the GRU had about $\frac{3}{4}$. This made the GRU a good choice for the recurrent neural network architecture, as it offered one of the lowest mean errors while being faster to train and run than the LSTM.

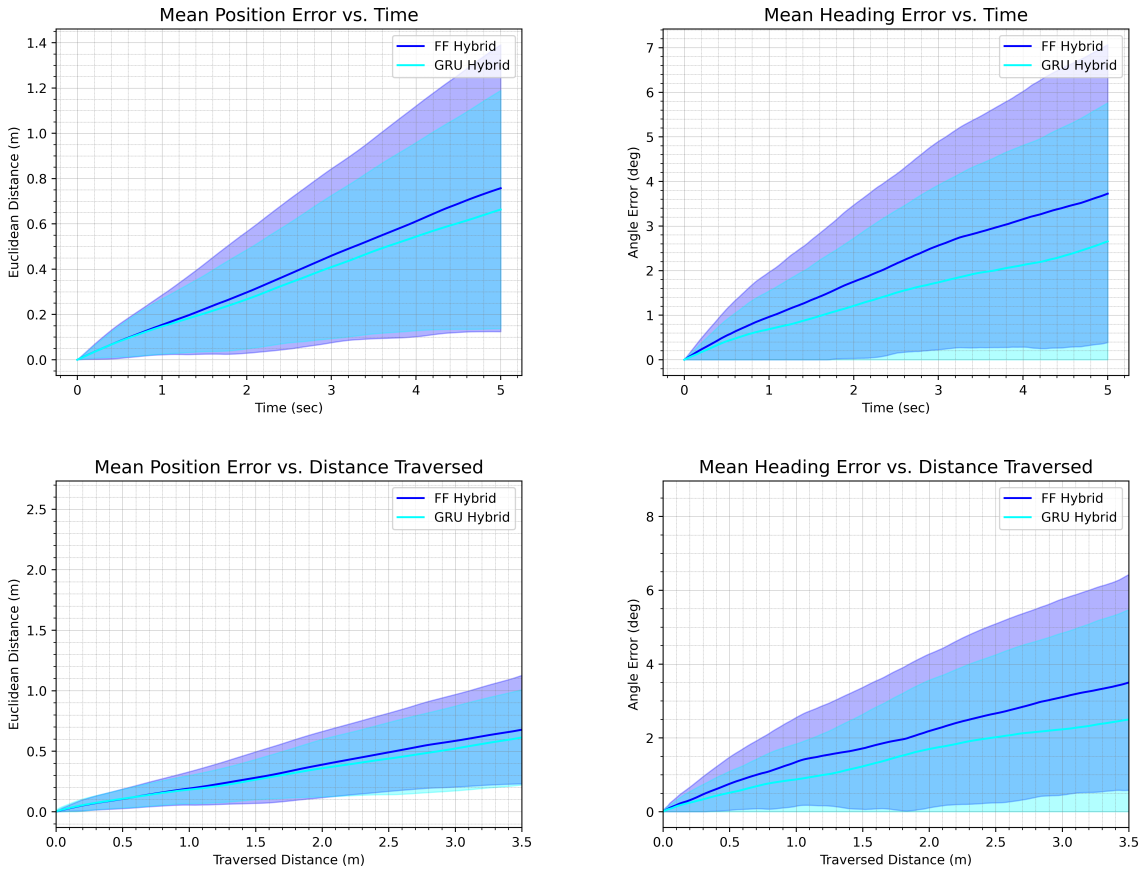


Figure 3.44: Comparison between the residuals-learning GRU hybrid model and the residuals-learning feedforward hybrid model for position & heading error over time & distance on the low speed test dataset

Figure 3.44 shows that the GRU hybrid model performs better than the feedforward hybrid model from the previous section (Figure 3.40). At the end of a 5 second prediction horizon, mean position error relative to the kinematic model has decreased by an extra 10%, while mean heading error has decreased an extra 12%. At 3.5 m traversed distance, mean position error has decreased an extra 7% and mean heading error has decreased an extra 20%.

3.6 Results and Conclusion

In summary, the tested models include the kinematic bicycle model, dynamic bicycle model, feedforward only neural network, feedforward hybrid model, and recurrent hybrid models. The hybrid models could either use their neural network block to learn vehicle frame velocities directly, or learn residuals for the velocities predicted by its submodels.

Between the kinematic model and the dynamic model (Figures 3.17, 3.18, 3.19, 3.20), both models had similar mean error on the low speed dataset (<3 m/s), however the kinematic model had a lower standard deviation in mean error. On the high speed dataset, the dynamic model performed better, as wheel slip starts to become non-negligible. However for our usecase, we were concerned with high accuracy maneuvering at low speeds where wheel slip is negligible, so decided to use the kinematic model as it was simpler to run.

Between feedforward and recurrent neural network blocks for the hybrid models, the recurrent based models were more accurate (Figure 3.44). However the feedforward models were much faster to train and evaluate, since they considered predictions at different time steps to be independent so can train or run data in large batches concurrently. The recurrent network based models need to run data sequentially as evaluations at each time step are dependent on the hidden state from the preceding evaluation. On an i7-6560U CPU, the feedforward network averages about 15 us per time step, while the recurrent network averages about 100 us (on top of time needed for the other blocks in the hybrid model to run).

The purely feedforward neural network model was disregarded as it performed worse than the kinematic model on our data (Figure 3.35). This was likely because it needed to learn how a vehicle behaves from scratch, using limited training data, and did not consider inputs from past time steps for a system with significant delays.

Overall, we found that the best model we had was the hybrid model with a GRU neural network learning residuals (Figure 3.42 & 3.44). This model had 3 hidden layers with tanh activation, and 16, 16, & 12 units in each layer. While the same accuracy could be attained with LSTM cells instead (shown in Figure 3.43), GRU cells were chosen as they had fewer trainable parameters so were slightly faster to train and run. Although the RNN cells are even simpler and less computationally expensive to use, they weren't able to retain enough information over large time steps to model the delays in the physical system.

Figure 3.45 shows this model compared to the kinematic model. At the end of a 5 second prediction horizon, the GRU hybrid model's mean position error is 23 cm (26%) lower, and mean heading error is 3° (56%) lower. When predicting over 3.5 m traversed distance, final mean position error is 27 cm (30%) lower, and mean heading error is 3° (50%) lower.

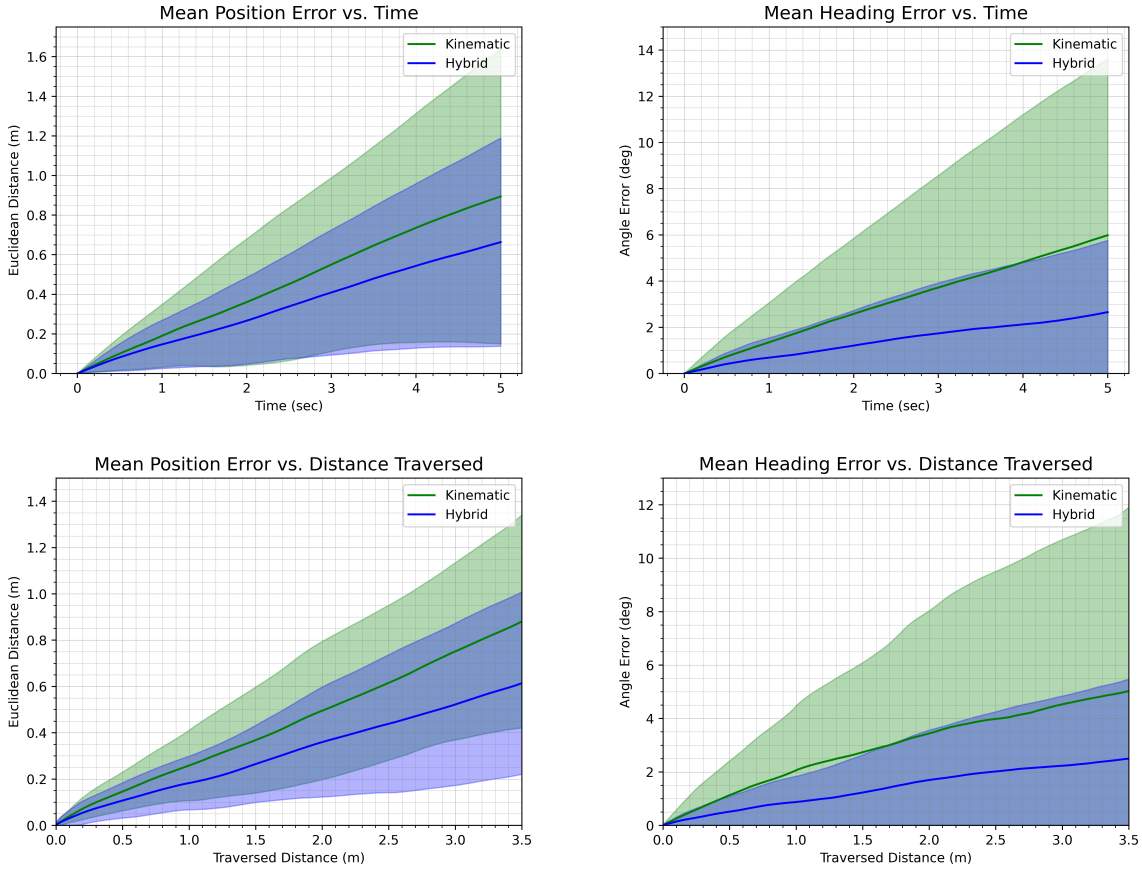


Figure 3.45: Comparison between the residuals-learning GRU hybrid model and the kinematic model for position & heading error over time & distance on the low speed test dataset

Figure 3.46 shows examples of trajectories from the low speed test dataset graphed with predicted trajectories from the kinematic bicycle model (green) and the GRU hybrid model (blue).

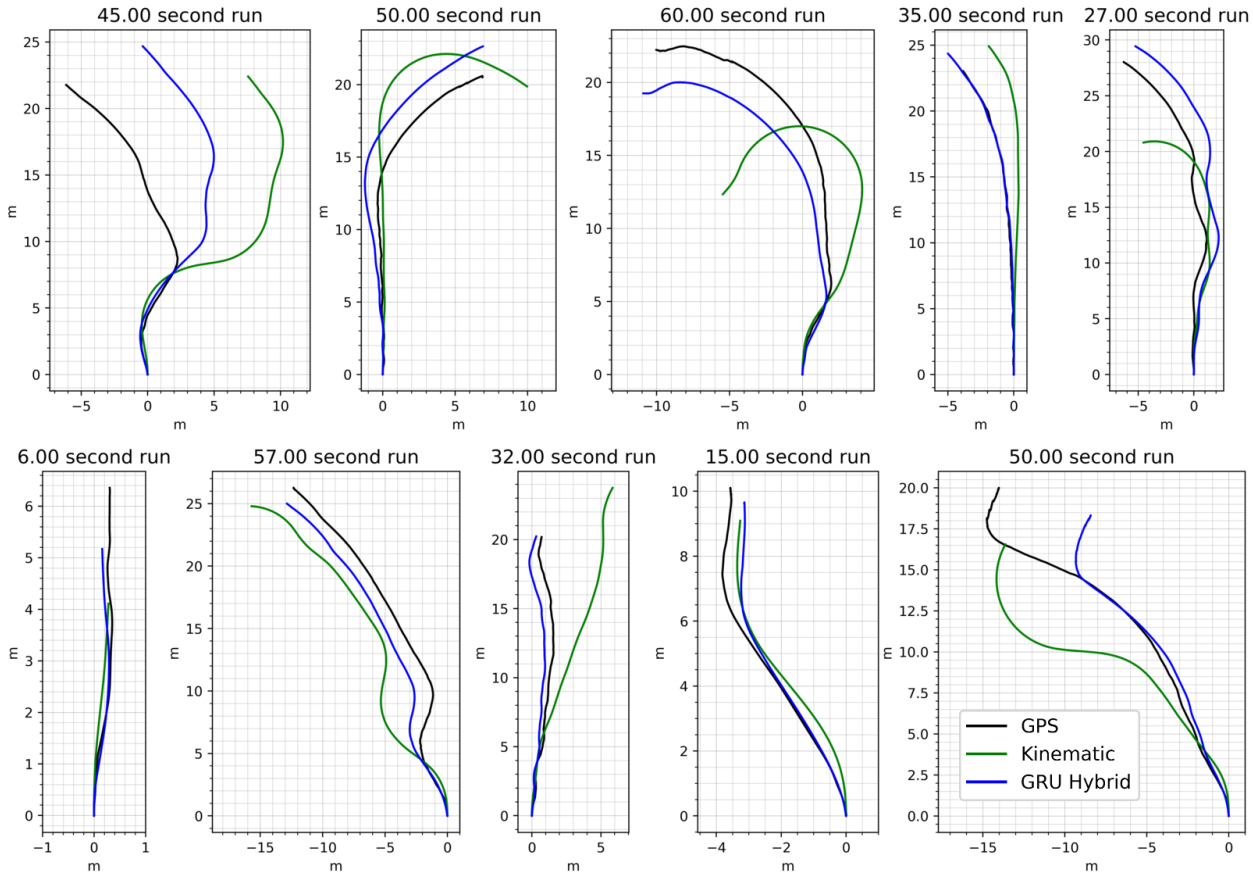


Figure 3.46: GPS and model predicted trajectories

Figure 3.47 shows model predictions of vehicle frame lateral velocity on a run from the low speed test dataset. The kinematic model always predicts 0 m/s due to the no slip assumption. The dynamic model performs better, predicting some lateral velocity, although there appears to be a time lag of about 1 second in the prediction. The GRU hybrid model using the kinematic model is able to learn the most accurate prediction of lateral velocity out of the evaluated models.

Figure 3.48 shows model predictions of vehicle yaw rate on the same run. Here the prediction from the kinematic model is more accurate than the prediction from the dynamic model, which still shows a 1 second time lag and doesn't predict the sharp peaks. Again, the GRU hybrid model using the kinematic model produced the most accurate estimate.

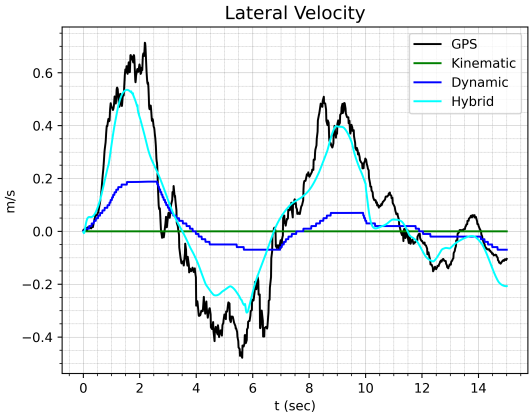


Figure 3.47: Comparison of models predicting lateral velocity

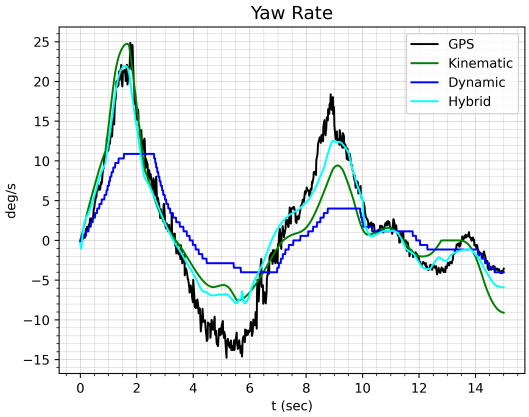
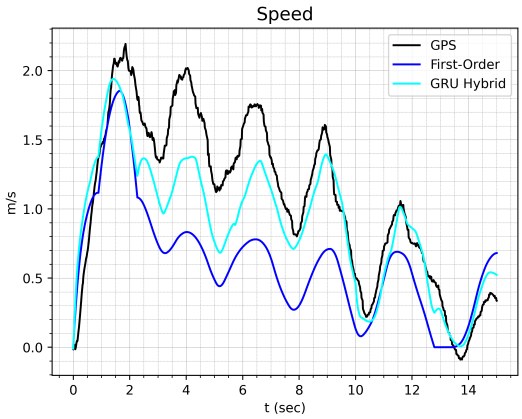


Figure 3.48: Comparison of models predicting yawrates

The hybrid model also predicts forward velocity, which can be compared against the first-order speed model incorporating the drivetrain that we used:



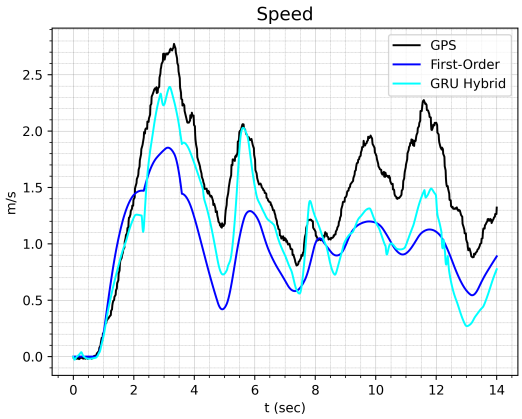


Figure 3.49: Comparison of models predicting forward speed for two example runs from the low speed test dataset

Overall, the hybrid model improves upon the first-order speed model, but the predictions were still not as accurate as the lateral and yaw velocity predictions. We found that the primary limiting factor in our overall model accuracy was the difficulty in predicting forward speed. As an example, Figure 3.50 compares the mean prediction error for a model that perfectly predicts forward speed but uses the GRU hybrid model for lateral velocity and yawrate, versus a model that perfectly predicts lateral velocity and yawrate but uses the GRU hybrid model to predict forward speed:

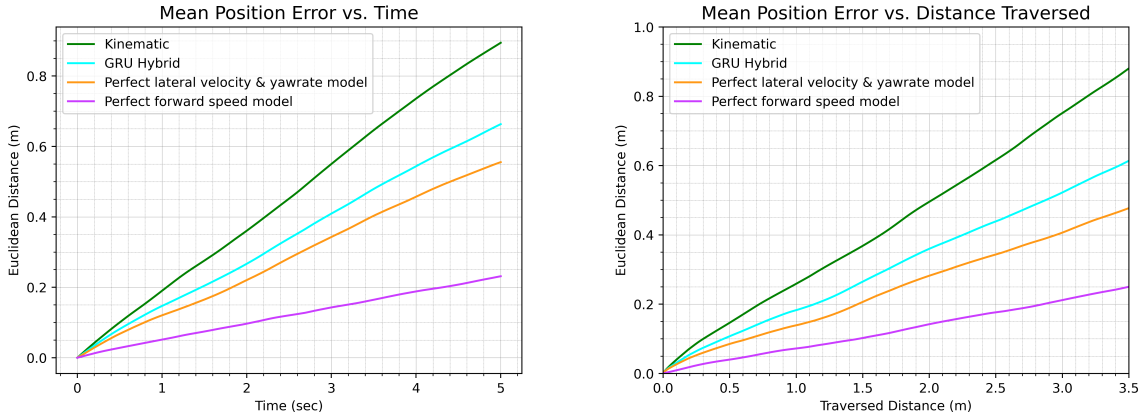


Figure 3.50: Comparison of models with perfect modelling of forward speed or lateral and yaw velocities

In the plot of error over time (Figure 3.50, left), we can see our most accurate model, the GRU hybrid model (cyan) with a 26% lower prediction error than the kinematic model at a prediction horizon of 5 seconds. If that model perfectly predicted lateral velocity and yawrate (orange), this prediction error would decrease to a 38% reduction over the kinematic model. If it predicted forward speed perfectly instead, we would see a reduction of 74% in error. This trend also appears when evaluating error over distance traversed.

In conclusion, our GRU + kinematic hybrid model shows an improvement in accuracy over all other models evaluated. However it was found that the primary limiting factor in modelling accuracy was in predicting forward speed. This is likely because the forward speed of the vehicle is dependent on additional variables on top of throttle position which are not considered. Terrain geometry for example is one variable that would greatly influence the forward motion of the vehicle especially at low speeds, and is one of the main reasons the speed oscillates about the controller setpoint so much. While we did not have enough data on terrain geometry to incorporate it into our model, if enough data was collected that we could account for it and improve our speed model, we would see significant improvement in our overall vehicle model compared to if we were able to perfectly model lateral and yaw velocities.

Chapter 4

Planning & Control

4.1 Introduction

In the previous chapter, we discussed methods to predict vehicle motion given a sequence of input commands. In this chapter, we briefly touch on the task of planning and control for the vehicle. When trying to move the vehicle from one point to another, for example in our problem of docking our vehicle to a payload, we want to do the inverse of our predictive model; we need a way to determine the sequence of commands that will move the vehicle to a desired location.

Unfortunately the problem is not as straightforward as forward modelling, as there are potentially many different sequences of input commands we could feed into our system that would achieve the same goal of moving the vehicle to a particular location. The problem can be thought of as a search problem, trying to find a sequence of commands from an infinite space of potential command sequences. Searching from the space of input commands allows us to use a forward model to predict where the vehicle will move, and evaluate whether or not to execute the command. The downside is that it adds a layer of abstraction to our search for a trajectory from the current state to our goal state. Alternatively, we can search for a sequence of states between our current state and our goal state, thus simplifying the problem of navigating around obstacles. However we still have the original problem of determining the commands needed to move from the vehicle from its current state to the next waypoint state.

Fortunately, there are many potential solutions to this problem, and we will discuss a few of these as well as explain the solution we used on our vehicle.

4.2 Related Work

One of Liu's [19] planners proposed was a model-based local planner using the Rapidly-exploring Random Tree (RRT) algorithm as a template. The RRT tree has nodes each representing a 6 DoF state, $q = [x, y, \theta, v_{forward}, v_{sliding}, \omega]^T$. To extend the RRT, a random state q_{random} is sampled, and a shooting method is used to determine commands which best extend the current state to q_{random}. Effectively this is achieved by randomly sampling control commands and durations then predicting resultant motion using a vehicle model. Parameters can be set by the user to dictate how much time is spent searching for appropriate input commands, and the threshold for error

below which the randomly sampled commands are accepted.

Hvamb [11] presents an alternative to this shooting method, instead using path smoothing rather than a model and random shooting to ensure a feasible path is generated for the vehicle controller. Hvamb's implementation is based on the work of Lekkas et al. [25]. In this paper, two methods of path smoothing are presented; one using Fermat's spirals, and the other using Euler spirals (also known as clothoids or Cornu spirals). The results of this paper show that while Euler spiral smoothing is simple to formulate, it is more computationally expensive to run due to the Fresnel integrals which do not have an analytical solution and must be solved numerically. On the other hand, Fermat's spiral is defined by much simpler parametric equations. The downside is that computing the length of a Fermat's spiral involves a Gaussian hypergeometric function, and care must also be taken to handle the speed singularity of the curve at the origin. Another consideration to make is the desired curvature profile as the vehicle traverses the path:

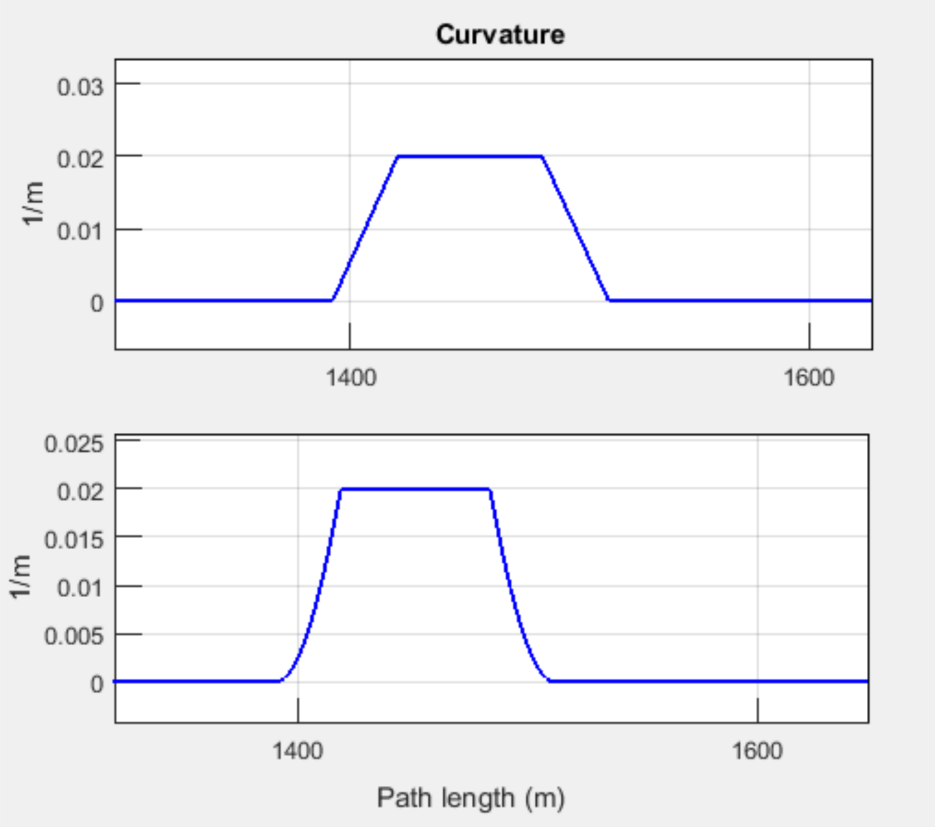


Figure 4.1: Closeup of curvature change for a Euler spiral (top) and a Fermat's spiral [11]

In Figure 4.8, we can see that the Euler's spiral (top) has a linearly changing curvature, while the Fermat's spiral's curvature (below) has an increasing rate of curvature change as the curvature increases. The advantage of the Fermat's spiral is smoother motion of the steering system. The disadvantage however is the vehicle is not turning as quickly as it could be, as the maximum rate of curvature change is only encountered near the end of a spiral transition segment.

4.3 Reference Path Generation

Suppose you have any planner that generates a sequence of waypoints between the vehicle's current location and its goal. From these waypoints, a reference path needs to be generated for a controller to follow. A simple naïve approach would be to just connect the points with straight line segments and hand it off to the controller. Despite the nonholonomic constraints on the vehicle restricting its motion (e.g. the vehicle cannot just rotate on the spot or slide laterally at-will), the controller would attempt to follow the path to the best of its abilities.

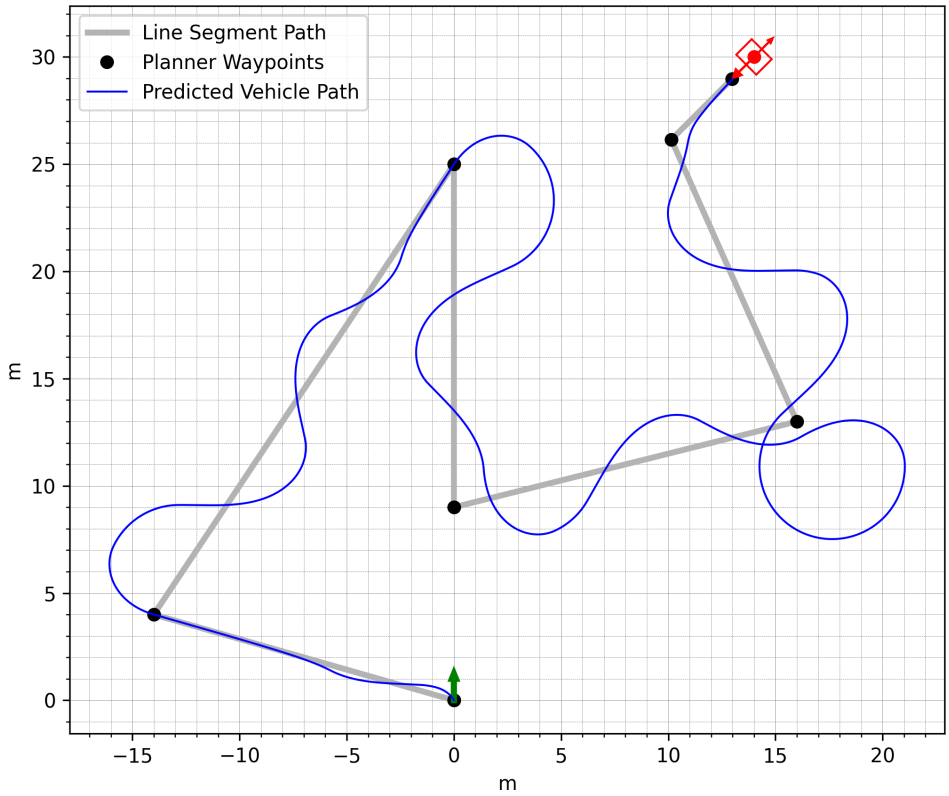


Figure 4.2: Control of a vehicle given a waypoints and straight line paths

Figure 4.2 shows a predicted vehicle path given a reference path of waypoints connected by straight line segments. While the controller does try to keep the vehicle on the path, there is significant overshoot leading to backtracking and large heading error when it does reach the goal position. This overshooting is undesirable, since there may be obstacles off the planned path that the vehicle could collide with. A method of generating a smooth reference path from these waypoints is needed to minimise or eliminate this deviation from the reference path.

The following method removes the curvature discontinuities found at the waypoints by introducing circular paths and Euler spiral transition segments. This work is based off the work of Hvamb [11], Lekkas et al. [25], and Chen et al. [26] on marine vehicles. In our implementation, by using power series approximations for the integration functions, and using pre-computed euler spiral segment templates, we were able to develop a functional reference path generator.

The first step is to place circular segments with radii greater than or equal to the vehicle's minimum turning radius at each waypoint. The position of the circles are dependent on the heading into and out of a waypoint, as shown below in Figure 4.3.

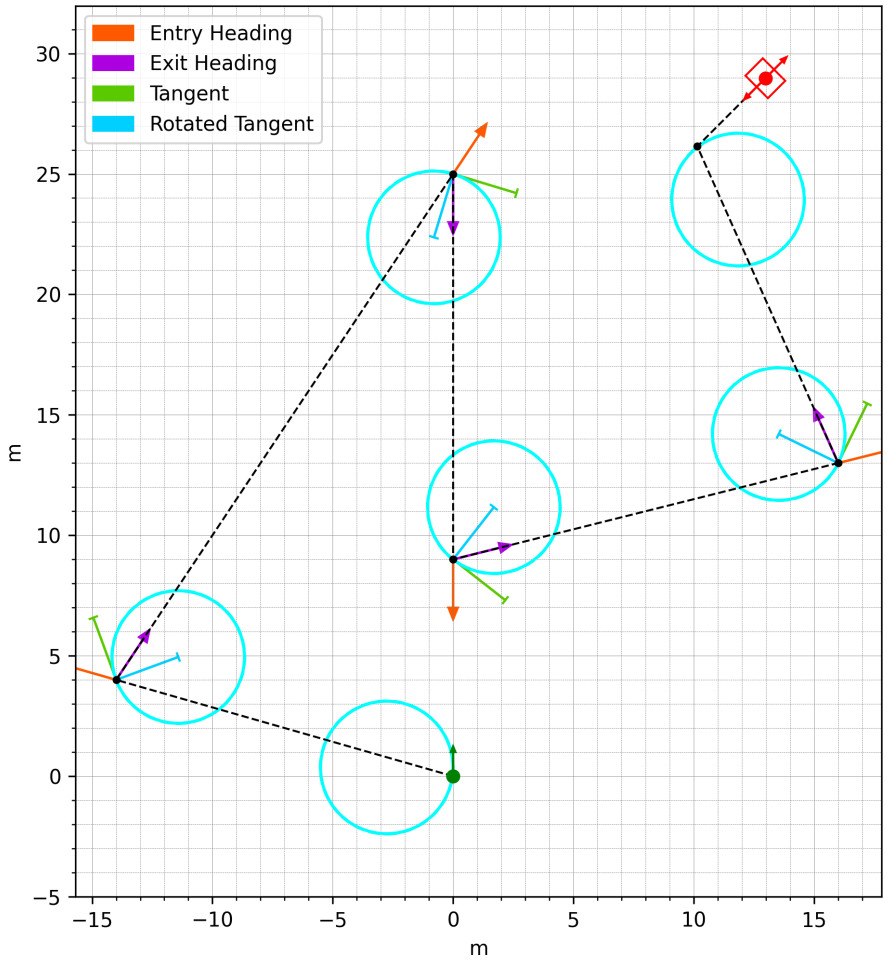


Figure 4.3: Entry and exit headings with tangents to position turning circles

For each waypoint (not including the first or last waypoints) we determine the heading into the waypoint and describe it with the unit vector v_{in} (orange arrow). Similarly the heading from the current waypoint to the next waypoint is determined and described with the unit vector v_{out} (purple arrow). The mean heading between these is used as the tangent v_{tangent} (green) for turning circle. Finally rotating the tangent vector 90° gives you the direction from the waypoint to the turning circle's centre. This vector is added to the waypoint and extended out by the circle radius

R.

$$v_{\text{in}}[i] = \frac{\text{waypoint}[i] - \text{waypoint}[i-1]}{\left|\text{waypoint}[i] - \text{waypoint}[i-1]\right|} \tag{4.1}$$

$$v_{\text{out}}[i] = \frac{\text{waypoint}[i+1] - \text{waypoint}[i]}{\left|\text{waypoint}[i+1] - \text{waypoint}[i]\right|} \tag{4.2}$$

$$v_{\text{tangent[i]}} = \frac{v_{\text{in}} + v_{\text{out}}}{\left|v_{\text{in}} + v_{\text{out}}\right|} \tag{4.3}$$

$$\text{rotate\_direction}[i] = -sgn\big((v_{\text{in},y} v_{\text{out},x}) - (v_{\text{in},x} v_{\text{out},y})\big) \tag{4.4}$$

$$\text{circle\_centre}[i] = (\text{rotate\_direction}[i]) \begin{bmatrix} -v_{\text{tangent},y} \\ v_{\text{tangent},x} \end{bmatrix} R + \text{waypoint}[i] \tag{4.5}$$

For now the first circle is placed such that the vehicle is tangent to it. This implies that the vehicle needs to start moving with the steering wheel already turned full lock. The final circle for now is placed such that the final approach from the last waypoint to the goal is tangent to it.

The next step is to connect the turning circles (cyan) with line segments (green). These line segments must join at a tangent to the turning circles, as shown below in Figure 4.4:

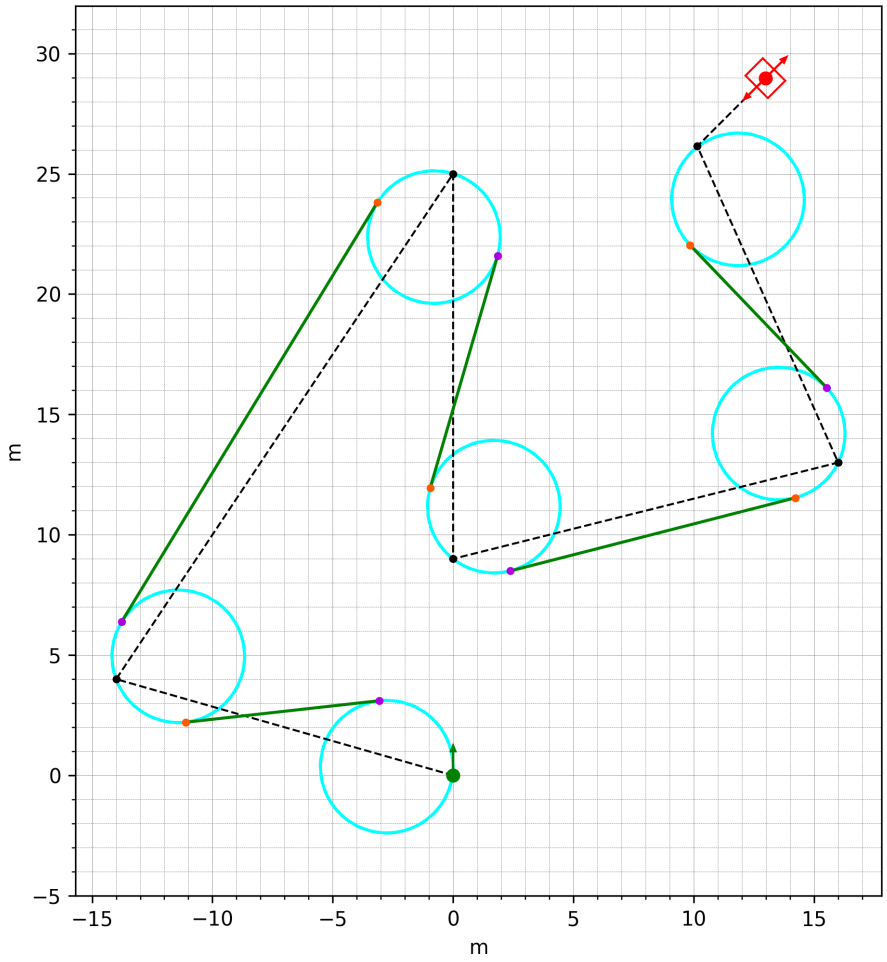


Figure 4.4: Line segments connecting between turning circles

The entry points (orange) and exit points (purple) to each turning circle can be calculated as follows:

If we exited from the previous circle travelling one direction, and are currently in the next circle travelling in the same direction (e.g. if we're going clockwise in both circles) i.e. if $\text{rotate\_direction}[i] = \text{rotate\_direction}[i-1]$:

First get the vector to the next circle centre:

$$v_{\text{line\_direction}} = \text{circle\_centre}[i] - \text{circle\_centre}[i-1] \tag{4.6}$$

Then get a vector perpendicular to $v_{\text{line\_direction}}$, and extend it by the circle radius R. This vector can then be used to calculate the coordinates for the exit point from the previous turning circle and the entry point to the current turning circle.

$$v_{\text{join}} = -\text{rotate\_direction}[i] \frac{R}{\left|v_{\text{line\_direction}}\right|} \begin{bmatrix} -v_{\text{next\_circle},y} \\ v_{\text{next\_circle},x} \end{bmatrix} \tag{4.7}$$

$$\text{exit\_point}[i] = v_{\text{join}} + \text{circle\_centre}[i-1] \tag{4.8}$$

$$\text{entry\_point}[i] = v_{\text{join}} + \text{circle\_centre}[i] \tag{4.9}$$

The calculations are slightly different for the case where we exited the previous turning circle in one direction, and are in the current turning circle which travels in the opposite direction (i.e. if $\text{rotate\_direction}[i] \neq \text{rotate\_direction}[i-1]$):

$$v'_{\text{line\_direction}} = \frac{v_{\text{line\_direction}}}{2} \tag{4.10}$$

$$\alpha = \arccos\left(\frac{R}{\left|v'_{\text{line\_direction}}\right|}\right) \tag{4.11}$$

Where R is the radius of the turning circle.
The orientation of $v'_{\text{line\_direction}}$ is found using the atan2 function:

$$\rho = atan2\left(v'_{\text{line\_direction},y}, v'_{\text{line\_direction},x}\right) \tag{4.12}$$

Then as before, we get the vector perpendicular to $v'_{\text{line\_direction}}$, extend by R, and use it to calculate the turning circle exit and entry point coordinates:

$$v_{\text{join}} = R \begin{bmatrix} \cos\left(\rho + \alpha(\text{rotate\_direction}[i])\right) \\ \sin\left(\rho + \alpha(\text{rotate\_direction}[i])\right) \end{bmatrix} \tag{4.13}$$

$$\text{exit\_point}[i] = v_{\text{join}} + \text{circle\_centre}[i-1] \tag{4.14}$$

$$\text{entry\_point}[i] = \text{circle\_centre}[i] - v_{\text{join}} \tag{4.15}$$

The coordinates along the line segments are found simply by interpolating between the entry and exit points on the turning circles.

At this point we already have a reference path that is much smoother than the original reference path made by simply connecting the waypoints with line segments. However, if we gave this reference path to our controller, the vehicle would probably behave similar to Figure 4.5:

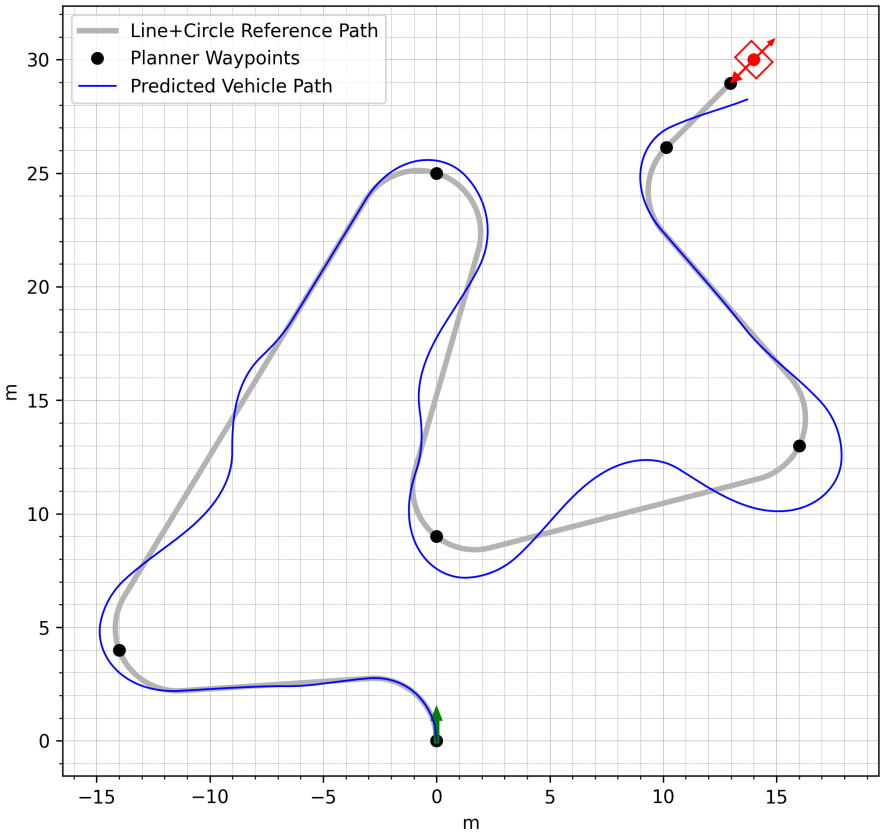


Figure 4.5: Control of a vehicle given a reference path of line and circle segments

While tracking of the reference path has greatly improved, the vehicle still overshoots at every corner. The reason is because transitioning from a line segment to circular segment requires an instantaneous change in steering angle, while our vehicle can only turn the steering wheel at a finite rate.

To reduce this error, we can take into account the vehicle's finite steering rate by using transition regions to smoothly transition curvature between straight line segments and the turning circles. There are many methods and types of curves that can do this, however here we'll present a method using the Euler spiral, whose curvature changes linearly over distance. To avoid having to calculate the Euler spiral for each time, we can generate templates ahead of time that are reused. Each template is defined by the rate of change in curvature, and the final curvature on the spiral. Thus if the vehicle is only turning the steering wheel at a fixed rate and all the turning circles have the same radius, we only need one template. Figure 4.6 below shows this transition segment (magenta):

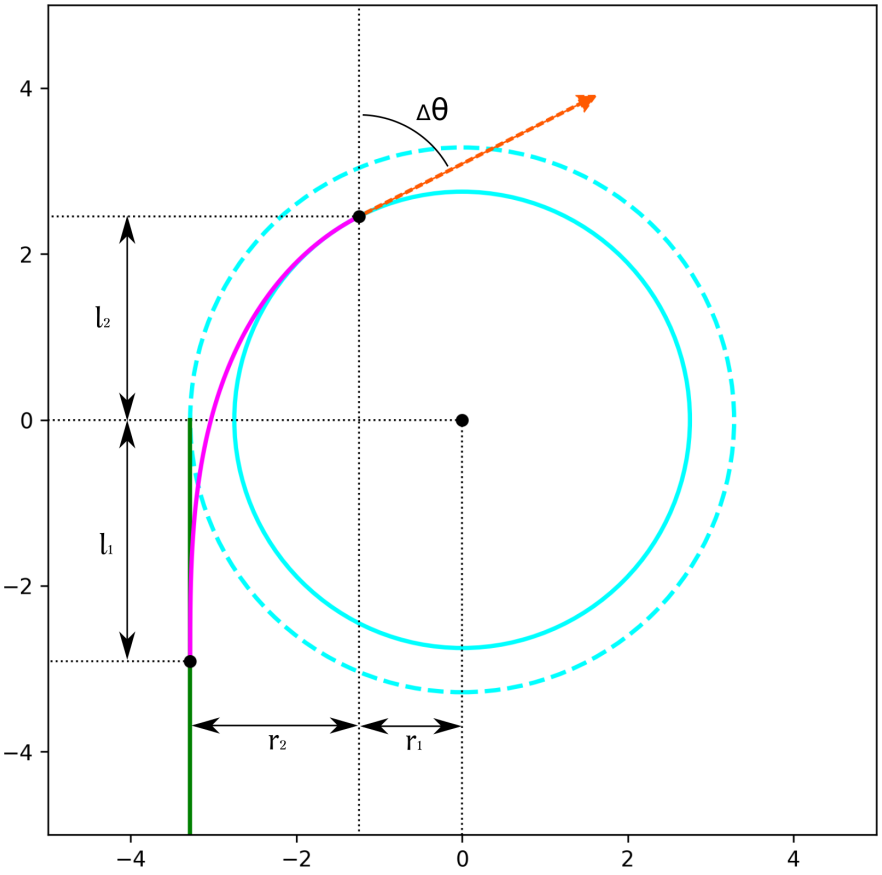


Figure 4.6: Euler spiral transition segment

The dimensions around the euler curve can be defined by the rate of curvature change we want along this curve $\Delta\kappa$, and the final curvature κ_{final}. In our case we set the $\Delta\kappa$ equal to the maximum curvature change rate the vehicle can achieve at 2 m/s, and ensure that the vehicle does not exceed this speed while travelling on any transition segments. For the final curvature, $\kappa_{\text{final}} = \frac{1}{\text{R}}$ where R is the minimum turning radius.

$$l_{\text{curve}} = \frac{\kappa_{\text{final}}}{\Delta\kappa} \tag{4.16}$$

$$\Delta\theta = \frac{1}{2}\Delta\kappa l_{\text{curve}}^2 \tag{4.17}$$

$$r_1 = R\cos\left(\Delta\theta\right) \tag{4.18}$$

r_2 involves calculating the normalised Fresnel Sine Integral over the length of the Euler spiral, but this can be approximated with a power series to M terms:

$$r_2 = \sqrt{\frac{\pi}{\Delta\kappa}}\int_0^t \sin\left(\frac{\pi}{2}s^2\right)ds \qquad \left(t = l_{\text{curve}}\sqrt{\tfrac{\Delta\kappa}{\pi}}\right)$$

$$r_2 \approx \sqrt{\frac{\pi}{\Delta\kappa}}\sum_{n=0}^{M}\left((-1)^n\left(\frac{\pi}{2}\right)^{2n+1}\frac{x^{4n+3}}{(2n+1)!(4n+3)}\right) \qquad \left(x = l_{\text{curve}}\sqrt{\tfrac{\Delta\kappa}{\pi}}\right)$$

The length of the Euler spiral parallel to the tangent line to the circle ($l_{\text{curve},y} = l_1 + l_2$) involves a normalised Fresnel Cosine Integral, which we also approximate with a power series.

$$l_{\text{curve},y} = \sqrt{\frac{\pi}{\Delta\kappa}} \int_0^t \cos\left(\frac{\pi}{2}s^2\right) ds \qquad \left(t = l_{\text{curve}}\sqrt{\tfrac{\Delta\kappa}{\pi}}\right)$$

$$l_{\text{curve},y} \approx \sqrt{\frac{\pi}{\Delta\kappa}} \sum_{n=0}^{M} \left((-1)^n \left(\frac{\pi}{2}\right)^{2n} \frac{x^{4n+1}}{(2n)!(4n+1)}\right) \qquad \left(x = l_{\text{curve}}\sqrt{\tfrac{\Delta\kappa}{\pi}}\right)$$

$$l_2 = R\sin\left(\Delta\theta\right) \tag{4.19}$$

$$l_1 = l_{\text{curve},y} - l_2 \tag{4.20}$$

With the dimensions around the Euler spiral known, the coordinates along the curve can be calculated using two Fresnel Integrals, where p is the distance along the curve from the straight end.

$$L' = \frac{p}{\sqrt{2Rl_{\text{curve}}}}$$

$$x_{\text{curve}} = \sqrt{2Rl_{\text{curve}}} \int_0^{L'} \cos(s^2) ds$$

$$x_{\text{curve}} \approx \sqrt{2Rl_{\text{curve}}} \sum_{n=0}^{M} \left((-1)^n \frac{L'^{4n+1}}{(2n)!(4n+1)}\right)$$

$$y_{\text{curve}} = \sqrt{2Rl_{\text{curve}}} \int_0^{L'} \sin(s^2) ds$$

$$y_{\text{curve}} \approx \sqrt{2Rl_{\text{curve}}} \sum_{n=0}^{M} \left((-1)^n \frac{L'^{4n+3}}{(2n+1)!(4n+3)}\right)$$

With these Euler curve segments to transition onto and off of turning circles, we also modify the position of the first and final turning circle. Assuming the vehicle begins at $(0,0)$ heading in the positive y direction (waypoint[0] is the initial position) we place the first turning circle such that the vehicle can begin with the steering wheel centred, and begin turning to enter the first circle:

$$\text{circle\_centre}[0] = \begin{bmatrix} sgn\left((r_1 + r_2)\text{waypoint}[1]_x\right) \\ l1 \end{bmatrix}$$

$$\text{rotate\_direction}[0] = -sgn\left(\text{waypoint}[1]_x\right)$$

For the final turning circle, we need a penultimate waypoint$[N-2]$ (N is the number of waypoints including the initial and goal points) that is offset from the goal position such that the final approach to the goal is along the same heading as the goal heading θ_{goal}. This allows the vehicle to straight out before docking with the payload.

$$\text{circle\_centre}[N-2] = \begin{bmatrix} \cos\theta_{\text{goal}} & -\sin\theta_{\text{goal}} \\ \sin\theta_{\text{goal}} & \cos\theta_{\text{goal}} \end{bmatrix} \begin{bmatrix} sgn\left((r_1 + r_2)\text{waypoint}[N-2]_x\right) \\ -l_1 \end{bmatrix} + \begin{bmatrix} \text{waypoint}[N-2]_x \\ \text{waypoint}[N-2]_y \end{bmatrix}$$

$$\text{rotate\_direction}[N-2] = -sgn\left(\text{waypoint}[N-2]_x\right)$$

To position the Euler curves, we use the Equations 4.6 to 4.15 to calculate the entry and exit points into a circle of radius $r_1 + r_2$, the transition circle, for each waypoint. These correspond to position $\begin{bmatrix} -(r_1+r_2) \\ 0 \end{bmatrix}$ in Figure 4.6, and we can apply a transformation based on the relative difference between these two poses for the curve.

The final resulting path is shown below, and can be followed perfectly by our vehicle model:

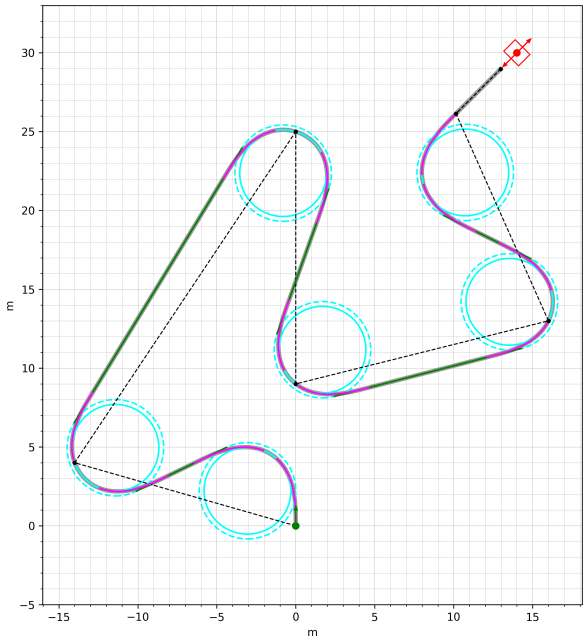


Figure 4.7: Final reference path with continuous curvature

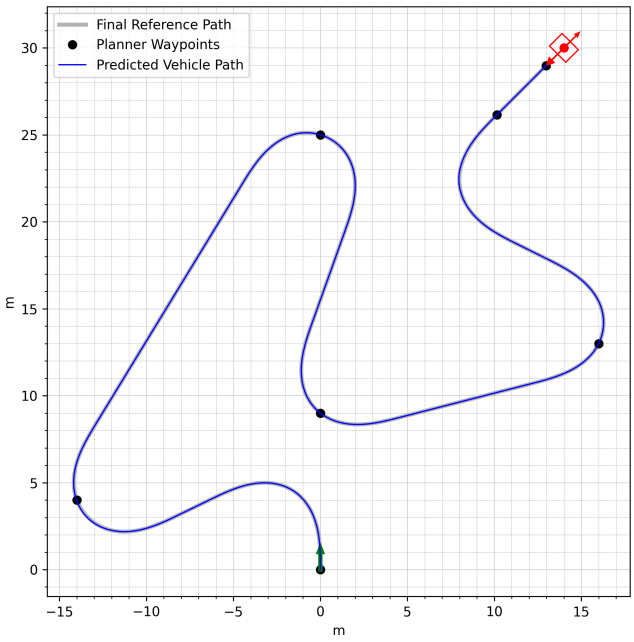


Figure 4.8: Control of a vehicle on the reference path from Figure 4.7

4.4 Results and Conclusion

Our implementation was able to generate kinematically feasible paths that the kinematic vehicle model could follow. For the power series approximations of the Fresnel Integrals, we found that as few as two terms in each approximation was still sufficient to produce a smooth final path (Figure 4.8).

Using Euler spirals instead of Fermat's spirals allows us to plan slightly sharper turns. However, one method we didn't touch on was increasing the maximum rate of curvature change ($\Delta\kappa$) the vehicle can track by reducing its speed. If we're using Euler's spirals, we can simply tune $\Delta\kappa$ to ensure a vehicle travelling at a constant speed, turning its steering wheel at its maximum turning rate, can traverse the path without deviation. If we're using Fermat's spirals instead and travelling at a constant speed, $\Delta\kappa$ increases over the length of the spiral, and we'd have to tune the spiral such that the final $\Delta\kappa$ is not too large for the vehicle to track. However, by reducing the vehicle's speed over the length of the spiral, we can increase the average $\Delta\kappa$ to be similar to an equivalent Euler spiral while still ensuring the path is kinematically feasible.

Overall, while our above implementation worked, it would be good to evaluate an implementation using Fermat's spiral smoothing in order to compare their relative performance.

Chapter 5

Conclusion and Future Work

In Chapter 2, we discussed design decisions made in the process of retrofitting the Yamaha UTV with actuators for drive by wire operation. This work also included development of controllers for forward speed and yaw rate commands.

For the yaw rate control system, we found that we may need to compensate for asymmetry in the vehicle's steering mechanism, as the vehicle was found to yaw anticlockwise faster than commanded, while it tracked clockwise yaw rate commands more accurately. Additionally, another potential improvement to this system could be the inclusion of feedback from the Xsens IMU or the NovAtel GNSS/INS. This could then correct any remaining error in the controller after it is tuned. Finally, the 0.5 second time delay is likely due to the finite turning speed of the steering wheel, and while increasing it should be possible, it would require upgrading either the motor controller or the 19V DC-DC converters. Furthermore, the steering wheel already turns faster than a human could, and increasing its speed further may put additional strain and possibly even damage the vehicle's steering system, or pose a hazard to a person sitting in the driver's seat.

The speed control system worked quite well, with only 5% maximum deviation from the setpoint while climbing a steep, rough slope. One critical shortcoming however was its poor performance at low speeds, under 2 m/s. We're unsure whether or not it is possible for the current controller to perform any better, given that it already outperforms human drivers. However incorporating point cloud data of the ground ahead of the vehicle may help improve performance, as local terrain geometry has a large impact on vehicle speed particular when travelling slowly. Additionally, improving controller performance at low speed may allow us to accurately model speed using speed commands rather than throttle commands, which need to take drivetrain dynamics into account in order to predict speed.

In Chapter 3, we explored different methods of modelling vehicle motion, concluding that a hybrid model which used a GRU recurrent neural network to compensate for errors in the conventional actuator and motion models achieved the best overall accuracy. In Figure 3.50, we showed that from our current best model, we gain a massive improvement in modelling accuracy if forward speed was modelled perfectly, compared to if both lateral velocity and yaw rate were modelled perfectly (74% versus 38% reduction in mean error compared to the kinematic model). We concluded that local terrain geometry likely has a large impact on the resulting speed of the vehicle, particularly at low speed. While we did not have enough data on local terrain geometry to incorporate it into our model, future work focused on this area may be able to incorporate point

cloud data of the ground in front of the vehicle to correct the predicted forward speed. Alternatively/additionally, the point cloud data could be used to improve the speed controller's low speed performance, after which we may be able to more accurately model speed using speed commands rather than throttle commands.

Finally in Chapter 4, we presented part of the method we used in our local navigation system; using Euler spiral segments to create reference paths with continuous curvature. The implementation worked well enough for our task, and using Euler spirals over Fermat's spirals allowed us to make plan sharper turns. However we could also implement a path smoothing algorithm using Fermat's spirals instead, and compare the two methods to determine whether our implementation of Euler spiral smoothing is still much more computationally expensive with the power series approximations we made to speed up the method. We could also plan forward speed as well, rather than simply capping it below a maximum value, which would allow us to adjust the vehicle path's maximum rate of curvature change without changing its steering wheel turning rate.

Bibliography

[1] MX-28T/R/AT/AR: Performance Graph. http://emanual.robotis.com/docs/en/dxl/mx/mx-28/#performance-graph. Online; accessed: 2020/06/01.

[2] Kairos Autonomi. *Pronto4 Steering Ring: Chain and Gear Assembly*, 2013. http://www.kairosautonomi.com/uploads/files/168/Pronto4-Steering-Ring-Chain-and-Gear-Assembly-0101.pdf.

[3] Kairos Autonomi. *Pronto4 Series 4 Installation Series*, 2014.

[4] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. pages 1094–1099, 06 2015.

[5] Neal Seegmiller, Forrest Rogers-Marcovitz, and Alonzo Kelly. Online calibration of vehicle powertrain and pose estimation parameters using integrated dynamics. In *Proceedings of IEEE International Conference on Robotics and Automation*, May 2012.

[6] Michael W. Bode. Learning the forward predictive model for an off-road skid-steer vehicle. Technical Report CMU-RI-TR-07-32, Carnegie Mellon University, Pittsburgh, PA, September 2007.

[7] Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauzá, Leslie Pack Kaelbling, Joshua B. Tenenbaum, and Alberto Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. *CoRR*, abs/1808.03246, 2018.

[8] Wikimedia Commons. Graph of cornering force vs slip angle, 2016. File: Tire Sip Angle.png.

[9] Wikimedia Commons. Magic formula curve, 2011. File: Magic Formula Curve.png.

[10] Yamaha Motor Company, Ltd. *Yamaha 2014 Viking Service Manual*, 2014. https://www.501parts.com/wp-content/uploads/2016/01/2014-2015_Yamaha_Viking_Service_Manual.pdf.

[11] Knut Hvamb. Motion planning algorithms for marine vehicles. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, July 2015.

[12] A. Jain, J. Guineau, C. Lim, W.P. Lincoln, M. Pomerantz, G. Sohl, and R. Steele. Roams: Planetary surface rover simulation environment. 08 2009.

[13] Neal Seegmiller. *Dynamic Model Formulation and Calibration for Wheeled Mobile Robots*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, October 2014.

[14] L. Ding, H. Gao, Z. Deng, P. Song, and R. Liu. Design of comprehensive high-fidelity/high-speed virtual simulation system for lunar rover. In *2008 IEEE Conference on Robotics, Automation and Mechatronics*, pages 1118–1123, 2008.

[15] D. Kim, H. Peng, S. Bai, and J. M. Maguire. Control of integrated powertrain with electronic throttle and automatic transmission. *IEEE Transactions on Control Systems Technology*, 15(3):474–482, 2007.

[16] Yamaha viking exhaust emission standards certification. https://ww3.arb.ca.gov/msprog/onroad/cert/ofhrv/ofmcatv_comply/2015/yamaha_sv_um0040262_686.pdf, 2015. Online; accessed: 2020/05/28.

[17] Cross The Road Electronics. *Talon SRX - User's Guide*, 2017. https://www.ctr-electronics.com/Talon%20SRX%20User's%20Guide.pdf.

[18] R. Rajamani. *Vehicle Dynamics and Control*. Mechanical Engineering Series. Springer US, 2005.

[19] Guan-Horng Liu. High dimensional planning and learning for off-road driving. Master's thesis, Carnegie Mellon University, Pittsburgh, PA, June 2017.

[20] E Bakker, L Nyborg, and H B Pacejka. Tyre modelling for use in vehicle dynamics studies.

[21] R. N. Jazar. *Vehicle Dynamics Theory and Application*. Springer US, 2008.

[22] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks. *arXiv e-prints*, page arXiv:1909.09586, September 2019.

[23] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv e-prints*, page arXiv:1412.3555, December 2014.

[24] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative Study of CNN and RNN for Natural Language Processing. *arXiv e-prints*, page arXiv:1702.01923, February 2017.

[25] Anastasios Lekkas, Andreas Reason Dahl, Morten Breivik, and Thor Fossen. Continuous-curvature path generation using fermat's spiral. *Modeling, Identification and Control (MIC)*, 34:183–198, 10 2013.

[26] X. Chen, J. Zhang, M. Yang, L. Zhong, and J. Dong. Continuous-curvature path generation using fermat's spiral for unmanned marine and aerial vehicles. In *2018 Chinese Control And Decision Conference (CCDC)*, pages 4911–4916, 2018.