Hardware report

AUTONOMOUS DRIFTING RC CAR USING REINFORCEMENT LEARNING

May 9, 2018

Supervisor: Dr. D. Schnieders
Sourav Bhattacharjee (3035123796)
Kanak Dipak Kabara (3035164221)
Rachit Jain (3035134721)

Written by Rachit Jain

Abstract

The advent of self-driving cars has pushed the boundaries on the safety of automobiles, but most modern self-driving car systems ignore the possibility of a car slipping resulting from inclement weather or driver error [1]. Passengers and bystanders would benefit heavily if self-driving cars could handle slipping by learning to drift with the turn rather than against it (by applying the brakes, or turning away, which is the instinctive action), preventing many fatalities [2].

Our project is aimed at studying the drifting of an autonomous remote controlled (RC) car using reinforcement learning (RL) techniques. Specifically, we experimented with a model-free approach with dueling double Deep Q-networks (DQN) and a model-based approach with Probabilistic Inference for Learning COntrol (PILCO) for finding an optimal drift controller. Since robotic systems are prone to wear with use, a simulator is used to model the car dynamics and train a preliminary drift controller which is then transferred to the real car.

Using these techniques, we were successful in obtaining an optimal drift controller on the simulator, which was stable and robust to varying physical conditions. Other than the drift controller, this project makes important contributions in the form of novel approaches like using DQN for obtaining a drift controller and using the policy learned from DQN for PILCO initialization. Additionally, this report presents a metric, D_m, to objectively quantify the quality of a sustained circular drift.

Acknowledgement

We would like to thank our supervisor, Dr. Dirk Schnieders, for guiding us throughout the project. We are also grateful for the help received from a friend, Mr. David Ip, who helped us acquire the hardware needed for this project. Finally, we are thankful for the help we received from Dr. Chris R. Roberts with various hardware issues encountered.

Contents

List of Tables

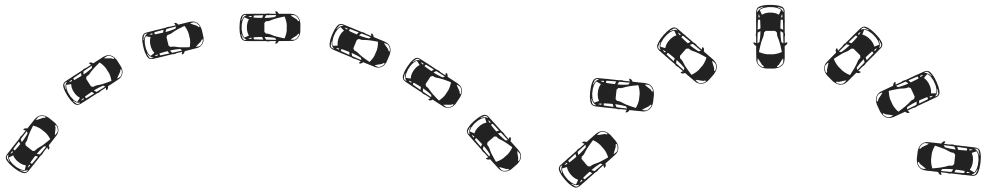
List of Figures

Abbreviations

Abbreviation	**Meaning**
BEC	Battery Eliminator Circuit
DQN	Deep Q-Networks
ESC	Electronic Speed Control
IMU	Inertial Measurement Unit
PILCO	Probabilistic Inference for Learning COntrol
RC car	Remote Control Car
RL	Reinforcement Learning
ROS	Robot Operating System

1 Introduction

Before discussing the implementation details of the project, it is crucial to understand the background of the problem we are trying to solve, and the actual scope. This section addresses that and highlights the need to study and solve the problem of drifting, and outlines how we plan to do so with an approach based on simulation aided reinforcement learning. For the purposes of this report, drifting is defined as the oversteering of a car which results in the loss of traction of the rear wheels. This results in the front wheels pointing in the opposite direction to the turn and the car appears to be moving sideways as shown in Figure 1.



(a) Steering around bend **(b) Drifting around bend**

Figure 1: Drifting is defined as the oversteering of a car which results in the loss of traction of the rear wheels. This results in the front wheels pointing in the opposite direction to the turn and the car appears to be moving sideways. The diagram illustrates the difference between simply turning around a bend and drifting around a bend.

1.1 Background and Motivation

Passenger vehicles usually implement stability control in a number of ways like differential braking [3], active steering [4] [5] or integrated chassis control [6] [7] [8]. Other methods, based on independent wheel torque, have also been developed to make passenger vehicles more stable. However, these methods function by making sure that the tires avoid slipping. In doing so, these methods essentially restrict the operation of the vehicle. Similarly, control algorithms in current self-driving car systems (Anti-lock brake systems, Electronic stability control etc.) try and mitigate the chances of slipping due to its unpredictable nature [1]. Sufficiently lowering the speed of the car and making turns that are not too tight will mostly

prevent slipping, but this does not consider cases where the system must make evasive moves to avoid crashes or when a car is already in a slipping state due to the driver's fault. For example, hydroplaning, which refers to a situation where a layer of water builds up between the car tires and the road, is a major reason for vehicle accidents. According to the United States' Department of Transportation, 15.97% of all vehicle crash fatalities in the United States [2] are attributed to wet and icy roads. An autonomous car system should be prepared for the scenarios outlined above to ensure the safety of the passenger and bystanders, regardless of the weather conditions or the state of the car. To reduce fatalities and ensure that these car systems are as robust and safe as possible, it is essential to study drifting, and eventually deduce how cars can respond quickly to unintentional slipping states as those encountered due to hydroplaning. Not only can drifting be useful to steer out of these unintentional slipping states, but can also be useful in taking full advantage of the capabilities of a vehicle to avoid accidents in emergencies.

Many of the systems discussed above try to tackle the issue of stability control and slipping by approaching it as an optimal control and open looped problem with explicit dynamics model. Approaches using optimal control are often deterministic and use closed-form expressible equations of motions. The resulting policies depends entirely on the model used to compute them. Sometimes, these restrictions on the model neglect parts of the true system either because they are non-linear or they are just not well-enough understood to be expressed in equations. We thus propose a method that does not rely on explicit equations of motion, but rather on an implicit understanding of the world obtained by trial and error.

Another motivation to study drifting is that Paul Frère [9] points out the usefulness of drifting to turn fast around sharp bends. Since high-speed stability is of greater importance to ordinary touring vehicles and competition cars, they tend to understeer, and for a competition car average circuit time is improved by going fast through fast bends while slowing down through sharp ones [9]. However, a car is able to turn sharp bends faster by drifting because the yaw angle formed by the drift brings the vehicle in line with the straight path following the bend even before the vehicle completes the turn [9].

1.2 Objective

The objective of this project is to get a remote controlled car to maintain a sustained circular drift autonomously. This paper proposes a framework for learning the best way to drift using simulation aided reinforcement learning which is one approach to solving the problem without having to input the dynamics of the system explicitly. Then the project aims to transfer the learned optimal drift policy or strategy from the simulation to a physical RC car for further learning.

1.3 Scope

The area of drifting falls into two categories – sustained drift and transient drift. Due to the wide breadth of the two categories and the time and cost constraints, our project will mainly focus on sustained drift, and more specifically steady state circular drift on an RC car with constant forward throttle. Additionally, despite the wide range of reinforcement learning algorithms available, due to reasons elaborated in the remaining of the report, we investigate two different algorithms to obtain the sustained circular drift controller - DQN and PILCO.

1.4 Deliverables

The complete implementation of the project is available on `https://github.com/kanakkabara/Autonomous-Drifting`. There are a few major deliverables in this project, which are outlined below:

1. Reinforcement Learning (RL) algorithms – Implementation of Double dueling Deep Q-networks for finding an optimal drift controller as well as model based policy search with PILCO.

2. Drift controller - A successful sustained circular drift controller along with tests to prove its robustness and stability.

3. Drift metric - A drift metric to objectively quantify the quality of a drift.

4. Simulator – We trained the RL algorithms on a simulated car that models the RC car in an environment with physics that mimic the real world. The

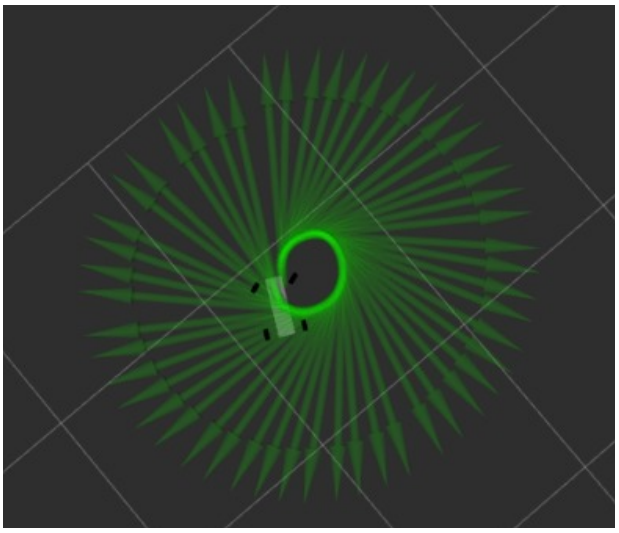


Figure 2: Time-lapsed path traced by the car on the simulator using the successful sustained circular drift controller.

environment was used to test and improve different RL algorithms efficiently and quickly, without causing any wear or damage to the RC car.

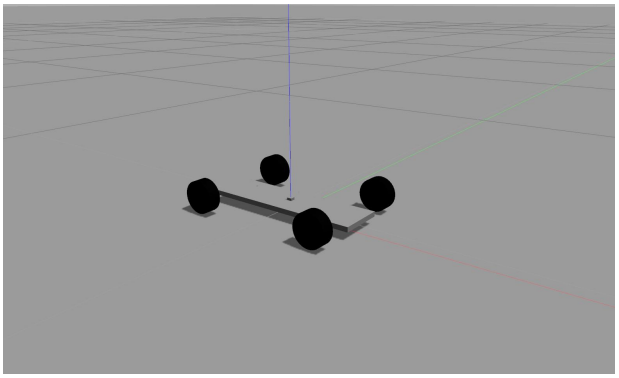


Figure 3: The final simulated car

5. A remote controlled (RC) car – This car is a 1/10th scale model of an actual car, integrated with sensors (Inertial measurement unit combined with magnetometer and optical sensors) for measuring data like the translational and angular velocities of the car to perform steady state circular drifting. The project then aims to transfer the optimal policy learned in simulator onto the RC car for validation.

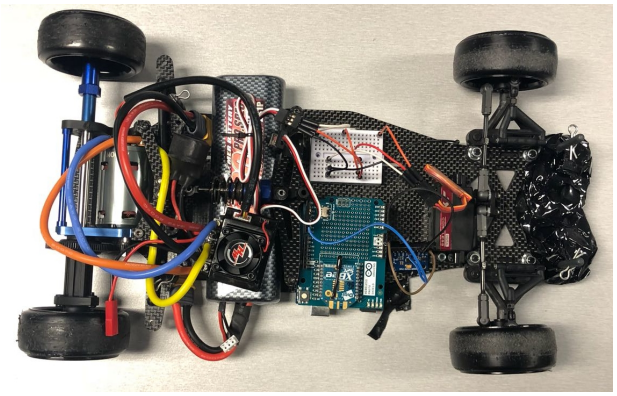


Figure 4: The final RC car

1.5 Contributions

The project introduces the following novel ideas, as elaborated further in *Reinforcement Learning Report*:

1. Using double dueling Deep Q-networks (DQN) to find an optimal drift controller.
2. Using policy learned from the DQN model to initialize PILCO learning.
3. A drift metric, D_m, to objectively evaluate a sustained circular drift:

$$D_m = \frac{1}{T}\sum_{t=0}^{T} \exp\left(-\frac{(||\boldsymbol{s}_t - \boldsymbol{s}_{target}||^2}{2\sigma^2}\right) \in [0,1]$$

1.6 Outline of Reports

The documentation for this project is divided into three reports. Although the reports share the same background and motivation behind the project, each emphasizes on the methodology, experiments, results and difficulties encountered for different aspects. A reader is thus suggested to refer to all three individual reports to acquire a complete understanding of the project. The three reports are as follows:

Report outlining the simulator and communication, written by Kanak Kabara, describes the implementation of an RC car in a simulated environment.

It also talks about the communication network connecting the various components of this project.

Report outlining the Reinforcement Learning Algorithms, written by Sourav Bhattacharjee, focuses on the main aspects of the project and contains the detailed description of the two Reinforcement Learning techniques used and the associated results.

This report highlights the implementation of the RC car and the various challenges faced in indoor localization and velocity estimation. The remainder of this report proceeds as follows. First, it will provide a literature review on the various methods that have been used to estimate the position of the RC car. Next, it will give a detailed description of all the different components involved in making the system. Finally, it will conclude by presenting some results and elaborating on the cost associated with the project.

2 Literature Review

2.1 Position Estimate

As mentioned in Section 3.1 of the *Reinforcement Learning Report*, we initially started with a full Markovian state representation of $[x,\ y,\ \theta,\ \dot{x},\ \dot{y},\ \dot{\theta}]$. In order to get an accurate estimate of the position of the car without using any external systems, various techniques for indoor localization were considered. A GPS is the obvious choice for a standalone car, however, a GPS is accurate to 3 meters at most [10], which is unacceptable for this project. Thus, other techniques were considered as summarized in the following paragraphs.

2.1.1 Wheel Encoder

One of the approaches we considered to get the position estimate of the car was using a wheel encoder. A wheel encoder is an electro-mechanical device that measures the rotations per minute (RPM) of the wheel of an RC car. The rotations obtained can then be multiplied by the circumference of the wheel to obtain the distance travelled by the RC car.

However, a major drawback of using a wheel encoder is the large error in the estimate of the distance travelled that amounts from the slipping of the tires [11]. In addition, as mentioned earlier, our state consists of x and y displacements but only a scalar distance can be obtained from the wheel encoder and not the required individual displacement components.

2.1.2 Computer Vision

One of the other methods to calculate the displacement of the RC car is to use computer vision. However, calculating displacement using computer vision and synchronizing it with the Inertial Measurement Unit (IMU) data is a challenging task. Furthermore, we wanted a stand-alone car, whereas using computer vision to calculate the displacement required external cameras.

3 Methodology

The Figure 5 shows the components of the RC car connected together. The subsequent sections describe these components and summarize the problems faced while using them.

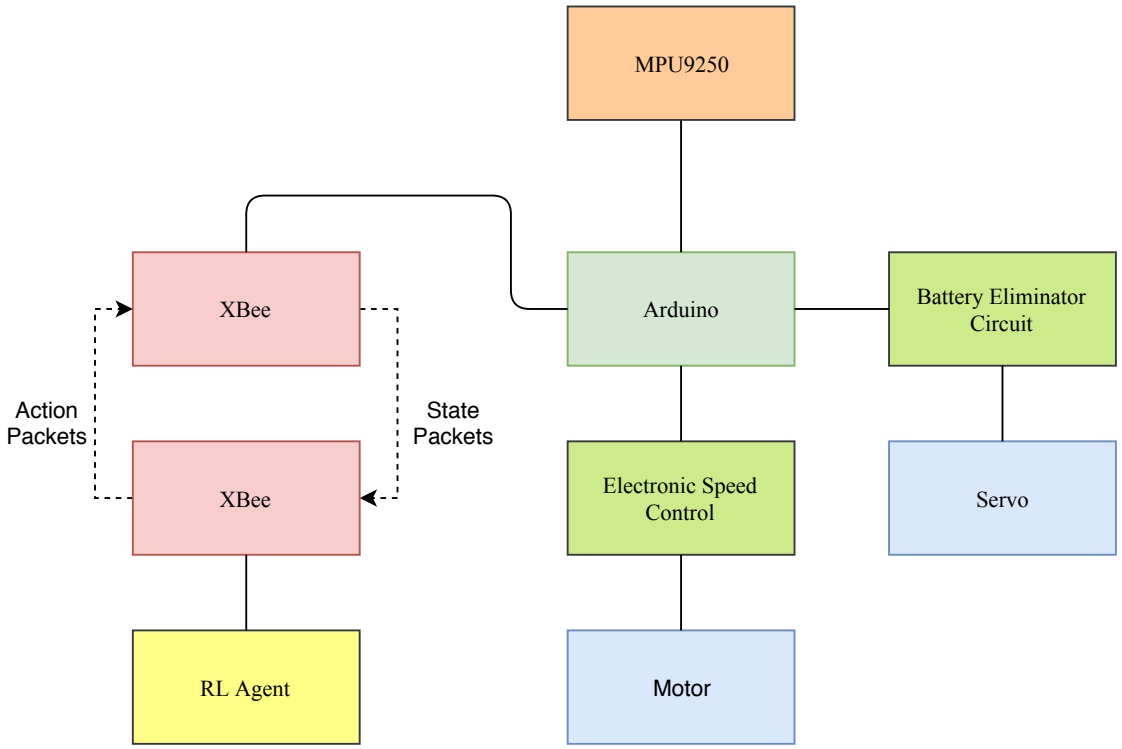
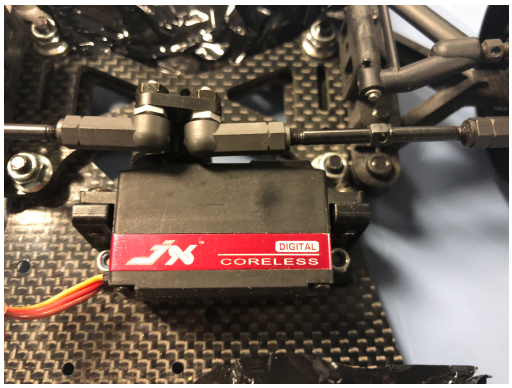


Figure 5: Components of RC car: Actuators are represented in blue and the sensor is represented in orange

3.1 Actuators

The RC car contains two actuators - a JX Coreless Servo and a Speed Passion 10.5R brushless motor. The servo controls the steering angle of the car while the motor is responsible for the throttle. The servo was constrained to an angle of 65 to 115 degrees from the horizontal axis, considering 90 degrees as the centre. This was done to make sure that the servo is not damaged and the values are predicated on the constraints of the axle in our RC car.



(a) JX Servo Coreless Digital Standard 4410MG

(b) Speed Passion 10.5R Brushless Motor

Figure 6: The two actuators used in the project - motor and servo

In order to eliminate the instability of the motor throughput resulting from voltage surges, we also included a battery eliminator circuit (BEC) for each actuator on the car.

3.2 Sensors

As mentioned in Section 3.1 of the *Reinforcement Learning Report*, we started with a full Markovian state representation for the car ($[x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}]$), and the subsequent sections outline some of the approaches undertaken to get estimates for the different components of the state.

3.2.1 Position Estimate

This section talks about the methods that were experimented with, to estimate the position of the car.

3.2.1.1 IMU double integration

The displacement of an object can be found by integrating the acceleration data twice (double integration). An IMU device fitted on the car provides the x, y and z components of the acceleration with respect to the car reference frame. We integrated the x and y components of the acceleration to get the x and y velocity of

the car. During integration, we assume the acceleration is constant for a very small time so integration was done every microsecond. The x and y velocities obtained were further integrated to find the x and y displacements of the car, which were then used to obtain an estimate for the car's position.

There was some drift and noise in the acceleration data from the IMU device and thus, a method called progressive averaging [12] was used to reduce the error in the acceleration data.

After inspecting the displacement obtained by double integration of the acceleration, we realized that it was diverging quicker than what we had expected. The solution to the problem is fusion of the IMU data with one more source – usually a GPS. An accurate GPS is expensive to buy and did not fit into out cost constraints. Thus, we considered another method for position estimation - optical flow.

3.2.1.2 Optical Flow

Optical flow is defined as the change of light in the image due to relative motion between the object and the camera's sensor [13]. The subsequent section discusses two devices (PX4Flow and Optical Mouse sensor) based on the optical flow principle which we used to calculate the displacement of the RC car.

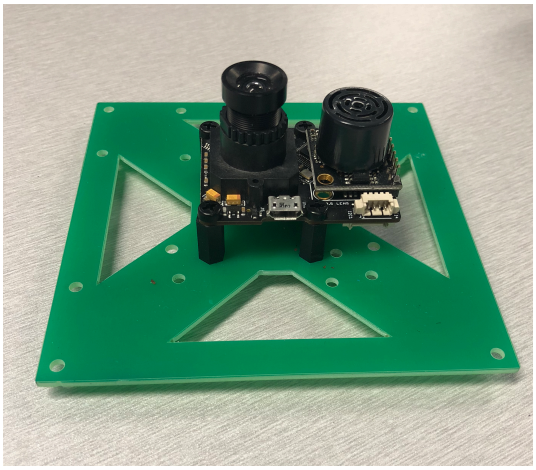


Figure 7: A PX4Flow camera fitted with 2mm lens. It works on the principle of optical flow and can be used to detect the velocity and displacement of a robot.

PX4Flow: We used an optical flow camera called PX4Flow for estimating the displacement of the RC car. For PX4Flow to work, it has to be mounted such that the lens of the camera focuses on some changing pattern. The camera

can be mounted on the RC car in two ways – facing down or facing up but both methods have their flaws.

If the camera is mounted facing upwards, the car is constrained to the environment which has some pattern at the height of at least 0.5 meters due to the focal length of lens being used [14]. We did not want the car to be constrained to some environment and thus, this approach was not taken.

Conversely, if the camera is mounted facing downwards, it requires a lens of focal length 2mm mounted at least 0.5 meters above the ground to measure the desired velocity of the car [14]. Mounting the camera at that height on the top of the car would result in an unstable structure and is not practical. Thus, we considered using another device using the optical flow principle - an optical mouse sensor.

Optical Mouse Sensor: Sekimori et al. [15] outline a method of utilizing the optical sensor in a mouse to calculate the displacement of mobile robots. This paper utilizes the same optical flow principles mentioned earlier, but in a much more accessible form, in the sense that significant patterns are not required to measure a discernible change.

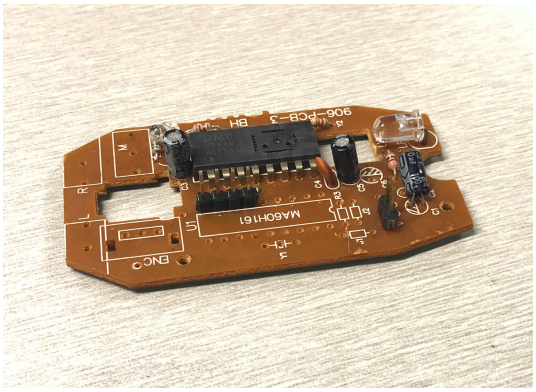


Figure 8: Optical Mouse Sensor: It works on the principle of optical flow but has its own flaws like it fails to detect high velocity movements.

In our project, we started by obtaining data from the optical sensor by connecting it to the Arduino. The data obtained from the mouse for smaller magnitudes of velocity was accurate, but there were major discrepancies in the data when the velocity was higher.

3.2.2 Velocity Estimation

As mentioned in Section 3.2.4 of the *Reinforcement Learning Report*, the state of the car was changed from

$$s_t = [x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}] \tag{1}$$

to

$$s_t = [\dot{x}_{car}, \dot{y}_{car}, \dot{\theta}] \tag{2}$$

For the new state, we needed the velocity of the car from the car's reference frame and several methods were tested as outlined below.

3.2.2.1 PX4Flow

Calculation of velocity was similar to the method of calculating distance using PX4Flow optical camera and we faced the same problems as discussed earlier.

3.2.2.2 IMU

The noise in the IMU acceleration data can be reduced by fusing it with one more data source. Fusion was done using a ROS library as discussed in Section 2.2.1 of the *Software Report*, which takes in data from different sources to obtain a better estimate for the acceleration. In our case, we used the IMU on an Android phone as the secondary data source.

3.3 Wireless Communication with XBee

Xbee is a device used for wireless communication between two devices and works on the popular IEEE 802.15.4 standard. A combination of the *xbee-python* library and ROS was used for end-to-end communication between two XBees.

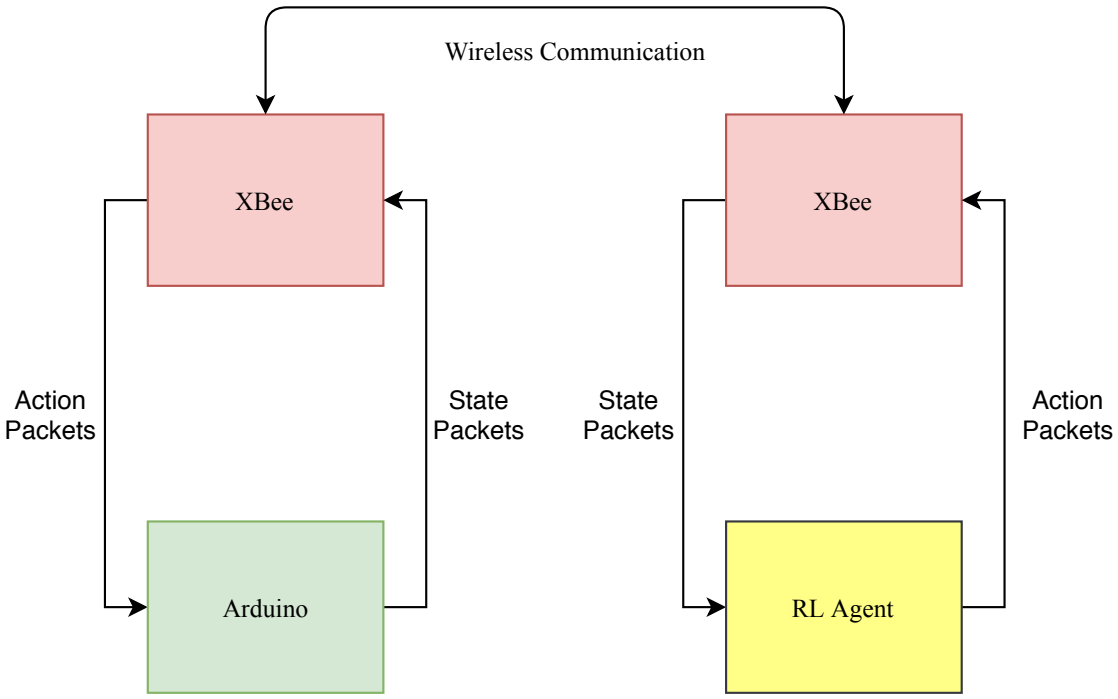


Figure 9: Transfer of action and state packets using XBees. The RL agent sends the action packets based on the state packets received from Arduino. The Arduino takes the actions based on action packets received from RL Agent and sends the corresponding state of the car.

Initially, we started with asynchronous communication between the Arduino and the RL Agent. This meant that the Arduino transferred messages pertaining to the car's state to the RL Agent regardless of whether an action message was received from the agent at every time step. As mentioned in Section 3.2.1 of the *Reinforcement Learning Report*, we need to correspond the action to the state so that we can model the dynamics of the car in the real world. Doing so was not possible with asynchronous communication. Additionally, the Arduino broadcasted the data at a faster rate than the IMU update rate, which led to duplicate packets being sent to RL Agent. Thus, we moved on to synchronous communication.

In synchronous communication between the RL Agent and the Arduino, the RL Agent waits for the state from the car before choosing the next action. We faced two problems with synchronous communication: loss of packets leading to

deadlock and mapping of state and actions.

Some action packets from the RL Agent to the Arduino were lost because of a buffer overflow, since an Arduino Uno uses only 100 bytes of buffer memory for receiving data. The data from the RL Agent was being broadcasted at a faster rate than Arduino could read, which resulted in packets being lost. To balance between a high throughput and solve the problem, we experimented by using different time delays on the RL Agent. The delay which solved the problem and achieved the highest throughput was 50 milliseconds.

Other than the packet loss due to buffer overflow, there were some packets lost because of a poor wireless connection between the RL Agent and the Arduino. To ensure packet delivery, we used a timeout which transmits the state packet if an action packet from the RL Agent is not received within a time window.

There was some latency between the action packets being received from RL Agent and the state packets being sent from the car. Thus, mapping of state and action was difficult. To correspond the actions taken to the state of the car, we used a time-stamp (epoch time) when sending the actions and state which solved the problem.

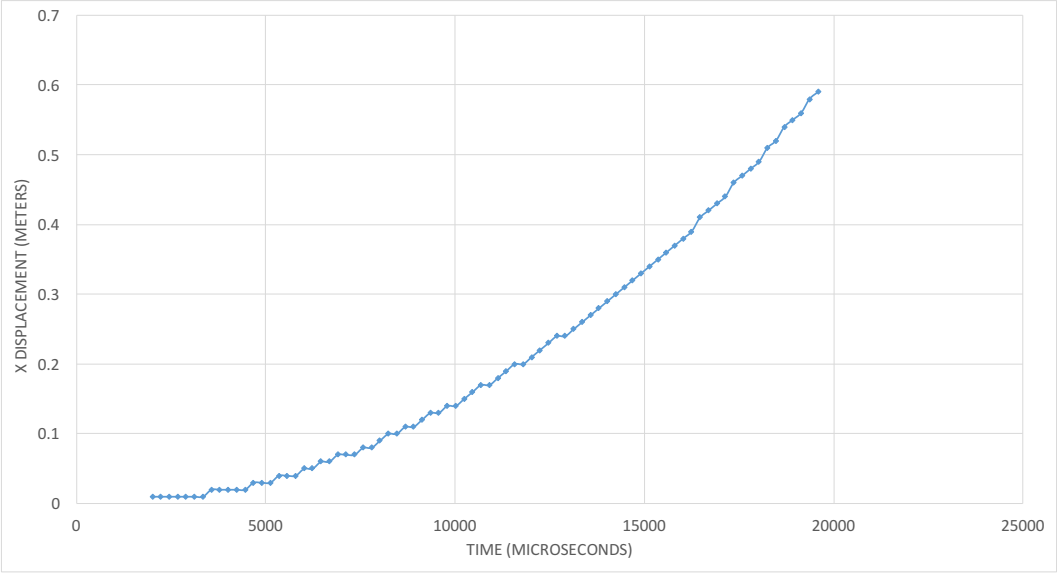
4 Results

The results were collected on a stationary car and are presented below.

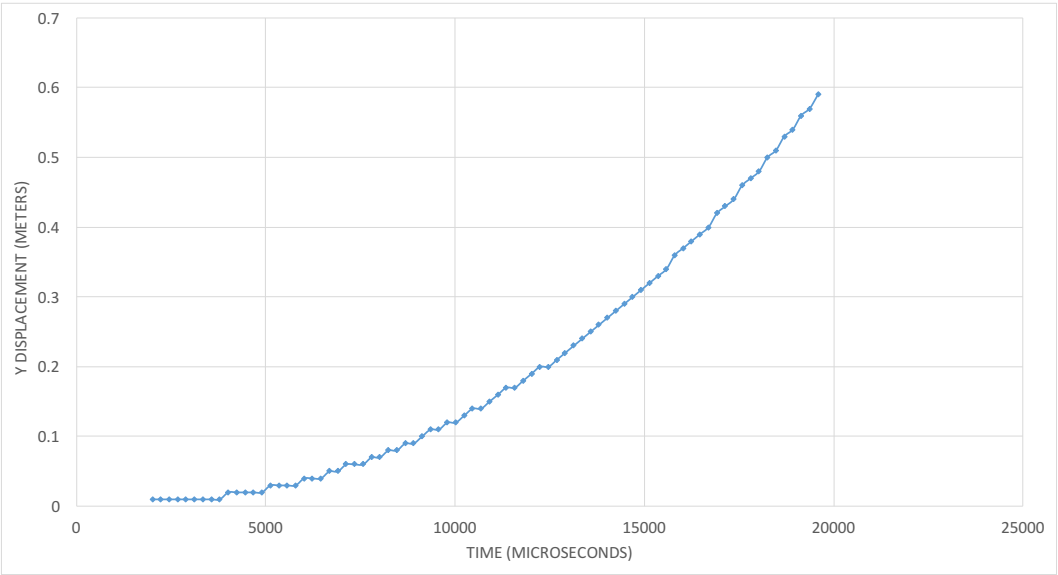
4.1 Position Estimate

4.1.1 IMU Double Integration

Figure 9 on the next page shows the error accumulation in the X and Y displacement after 20 seconds of the car being stationary. X or Y displacement and time are plotted on y-axis and x-axis respectively. The X and Y displacement has drifted around 0.6 meters after around 20 second of run.



(a) X Displacement



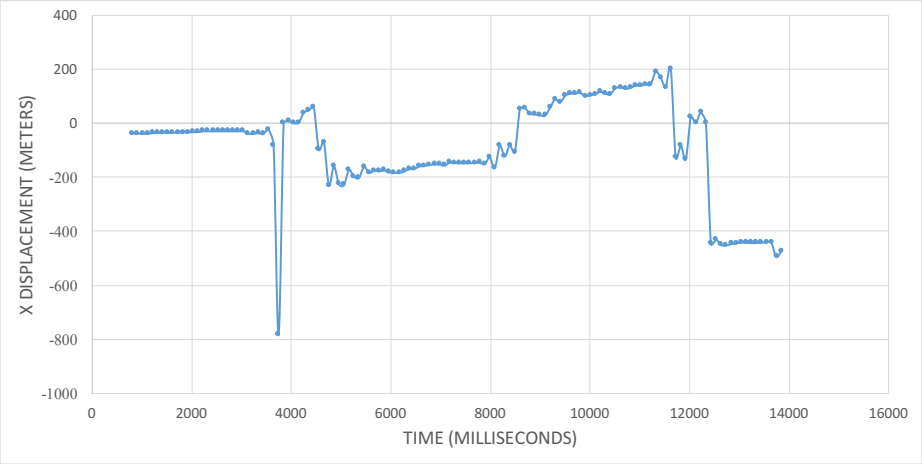
(b) Y Displacement

Figure 10: Displacement drift after integration: X or Y displacement and time are plotted on y-axis and x-axis respectively. The X and Y displacement has drifted around 0.6 meters after around 20 second of run.

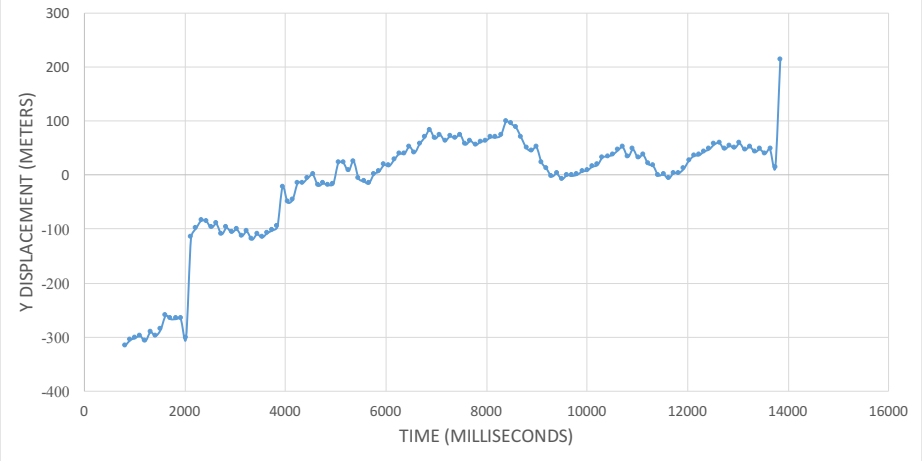
X and Y displacement has drifted around 0.6 meters in 20 seconds, which shows the problem of drift as mentioned earlier.

4.1.2 PX4Flow

The Figure 11 shows the drift in the X and Y displacement when plotted for 14 seconds. There was a drift of up to 800 meters in the X-displacement and 300 meters in Y-displacement of the car.



(a) X Displacement

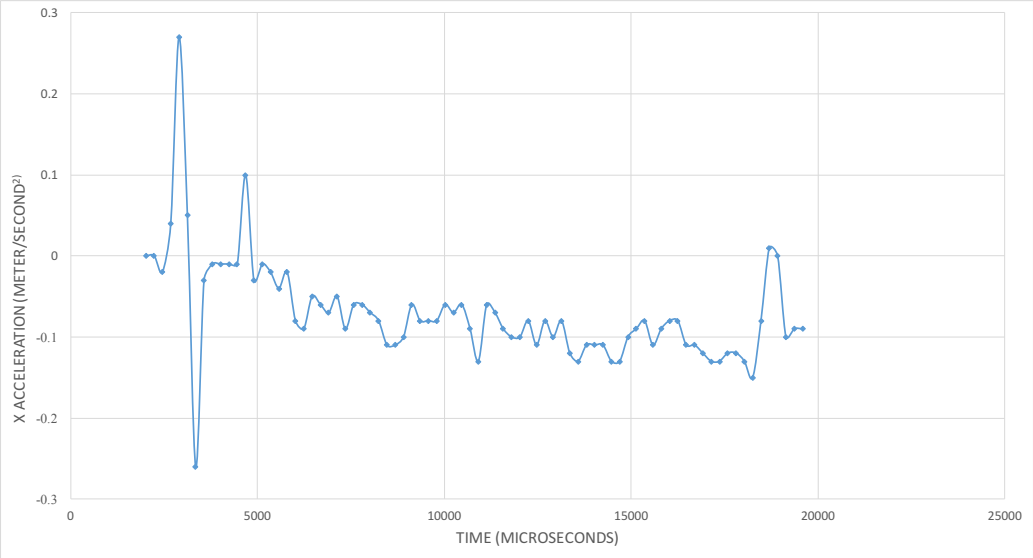


(b) Y Displacement

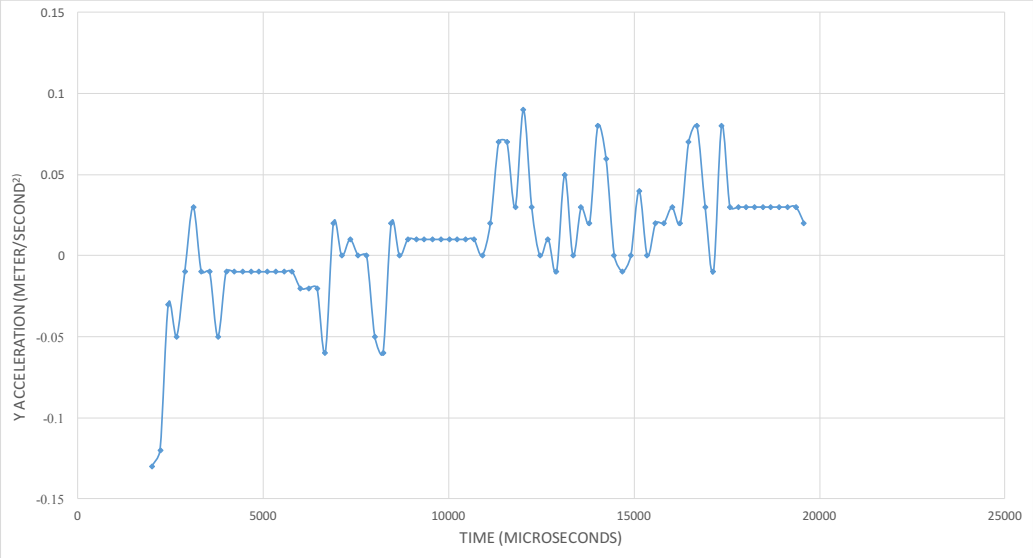
Figure 11: PX4Flow displacement drift: X or Y displacement and time are plotted on y-axis and x-axis respectively. X and Y displacement of the car has drifted around 800 and 300 meters respectively.

4.2 Madgwick Filters

We also used Madgwick filters to reduce the noise and error in the acceleration data. The Figure 12 shows the X and Y acceleration before the implementation of Madgwick filters.



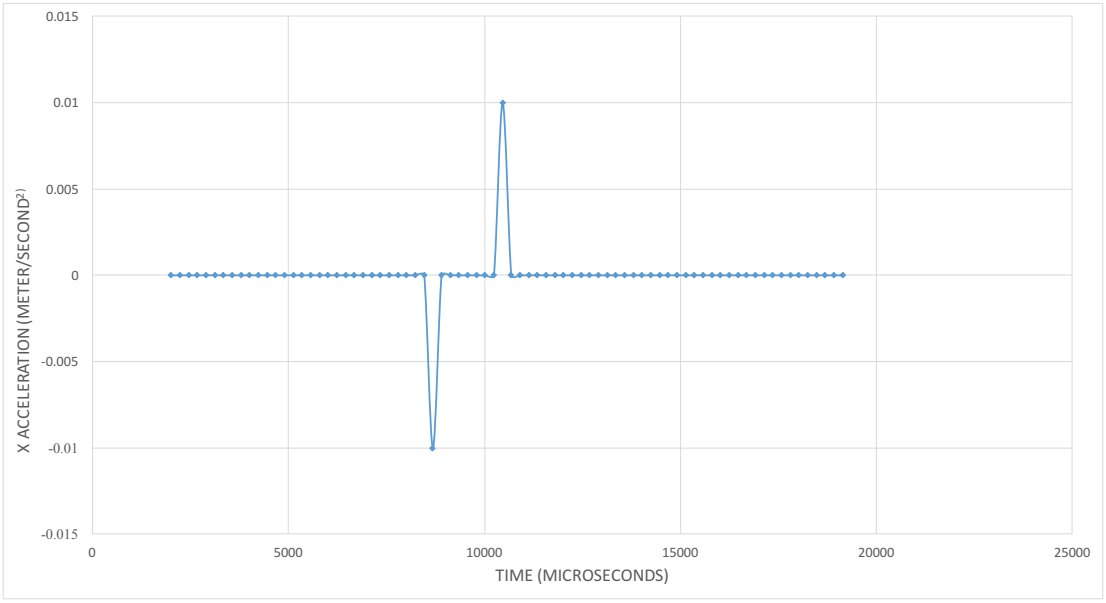
(a) X acceleration before filtering



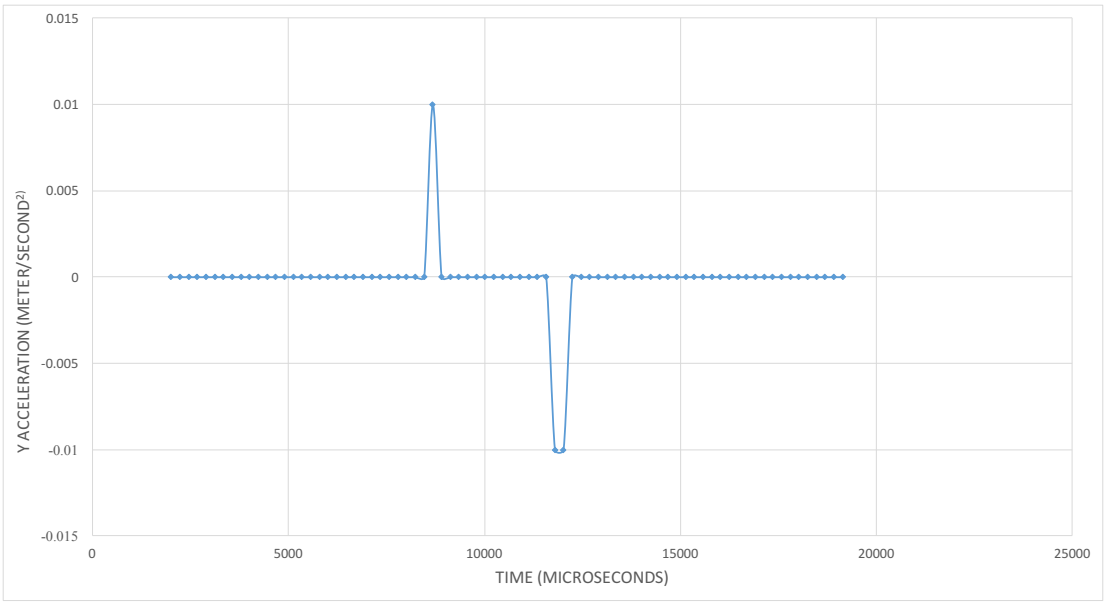
(b) Y acceleration before filtering

Figure 12: Before the use of Madgwick filters, there is a lot of noise in the acceleration data from the IMU.

After implementation of Madgwick filters, the X and Y acceleration of the car was stable as represented by Figure 13.



(a) X acceleration after filtering



(b) Y acceleration after filtering

Figure 13: After using Madgwick filters, the noise in X and Y acceleration has decreased drastically, giving us much more accurate data.

4.3 XBee Packet Drop

Table 1 summarizes the different experiments performed to alleviate the problems related to packet losses discussed earlier. In each experiment, 500 packets were sent from the RL Agent.

Experiment	Packets received by Arduino	Packets received (%)
Asynchronous	261	45.5%
Synchronous (10ms delay)	412	82.4%
Synchronous (30ms delay)	473	94.6%
Synchronous (50ms delay)	500	100%

Table 1: XBee synchronous and asynchronous experiments: Different delays were used in synchronous communication and the highest throughput was achieved with 50ms delay.

As mentioned earlier, adding a delay in synchronous communication solved the problem of packet loss.

4.4 Voltage Surges

As mentioned earlier, there were voltage surges when the motor was connected to the battery because of a single BEC. Figure 14 shows the voltage across the motor when only the steering command was sent.

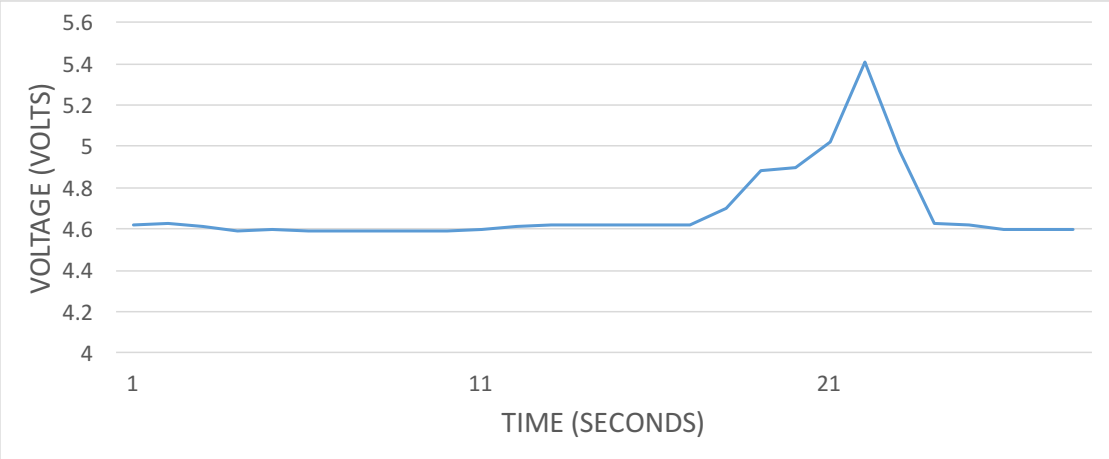


Figure 14: Voltage Surges without BEC: When the servo was given a command, there were voltage surges in motor which caused it to accelerate. This happened because of the use of one BEC.

5 Cost of Components

We realized that acquiring all the components of the car would not be possible in the given project budget. We were able to loan some of the parts of the RC car, mainly the chassis and servo. For the remaining parts, the prices are shown in Table 2.

Component	Price (in HK$)
Speed Passion 10.5R Brushless Motor	160
Arduino	160
XBee S2C and XBee Explorer	512
XBee Shield	150
MPU9250	146
Hobbywing XeRun 120A ESC	540
Drift Tires	290
Lipo Battery	180
Total	**2,138**

Table 2: Cost of all the components used in the car

6 Conclusion

To summarize, we justified why autonomous drifting cars are important and how drifting can be useful in emergencies to avoid accidents. As we have already discussed, current self driving cars and stability control techniques try to avoid slipping tires and in doing so, restrict the capability of the car. However, we need to exploit the full capability of a car during emergencies. So clearly, having an autonomous drifting car that learns an optimal drift control policy using our methods can help reduce the number of accidents caused by hydroplaning and make roads safer.

Motivated with this intention, we first discussed the different methods that we experimented with to estimate the position and velocity of the RC car. We also elaborated the synchronous and asynchronous communication between the Arduino and the RL Agent and the challenges we faced while implementing them. In addition, we presented the results for different experiments that were performed. Finally, we outlined the costs involved in assembling the RC car.

Although the initial aim of the project was to implement autonomous sustained circular drift in a physical RC car, we did not manage to achieve it completely owing mostly to hardware challenges associated with indoor localization and cost constraints (requiring a 4WD RC car). Nevertheless, much effort was put into closely modelling the physical properties of an RC car in the simulator as mentioned in Section 2.3.1 of the *Software Report*. Thus, given our success in finding a robust and stable sustained circular drift controller with the simulator, we firmly believe the results can be easily replicated on a physical RC car once the hardware is acquired.

References

[1] F. Zhang, J. Gonzales, K. Li, and F. Borrelli, "Autonomous drift cornering with mixed open-loop and closed-loop control," in *Proceedings IFAC World Congress*, 2017.

[2] S. Saha, P. Schramm, A. Nolan, and J. Hess, "Adverse weather conditions and fatal motor vehicle crashes in the united states, 1994-2012," *Environmental Health*, vol. 15, 2016.

[3] A. T. van Zanten, R. Erhardt, G. Landesfeind, and K. Pfaff, "Vehicle stabilization by the vehicle dynamics control system ESP," *IFAC Mechatronic Systems, Darmstadt, Germany*, pp. 95–102, 2000.

[4] J. Ackermann, "Robust control prevents car skidding," *IEEE Control Systems Magazine*, vol. 17, pp. 23–31, 1997.

[5] K. Yoshimoto, H. Tanaka, and S. Kawakami, "Proposal of driver assistance system for recovering vehicle stability from unstable states by automatic steering," in *Proceedings of the IEEE International Vehicle Electronics Conference*, 1999.

[6] A. Hac and M. Bodie, "Improvements in vehicle handling through integrated control of chassis systems," *International Journal of Vehicle Design*, vol. 29, no. 1, 2002.

[7] J. Wei, Y. Zhuoping, and Z. Lijun, "Integrated chassis control system for improving vehicle stability," in *Proceedings of the IEEE International Conference on Vehicular Electronics and Safety,*, 2006.

[8] A. Trachtler, "Integrated vehicle dynamics control using active brake steering and suspension systems," *International Journal of Vehicle Design*, vol. 36, no. 1, pp. 1–12, 2004.

[9] P. Frère, *Sports Car and Competition Driving*. Bentley, 1969.

[10] US Department of Defence, "Global positioning system standard positioning service performance standard," vol. 4, 2008.

[11] L. Ojeda, D. Cruz, G. Reina, and J. Borenstein, "Current-based slippage detection and odometry correction for mobile robots and planetary rovers," *IEEE TRANSACTIONS ON ROBOTICS*, vol. 22, 2006.

[12] K. Nirmala, A. G. Sreejitha, J. Mathewa, M. Sarpotdara, A. Suresha, A. Prakasha, M. Safonovaa, and J. Murthya, "Noise modeling and analysis of an IMU-based attitude sensor: Improvement of performance by filtering and sensor fusion," 2016. [Online]. Available: `https://arxiv.org/pdf/1608.07053.pdf`.

[13] B. K. Horn and B. G. Rhunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, [Online]. Available: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.66.562&rep=rep1&type=pdf`.

[14] D. Honegger, L. Meier, P. Tanskanen, and M. Pollefey, "An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications,"

[15] D. Sekimori and F. Miyazaki, "Precise dead-reckoning for mobile robots using multiple optical mouse sensors," [Online]. Available: `https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2009/ncr6_wjw27/ncr6_wjw27/docs/dead_reckoning_with_mouse_sensors.pdf`.