TurtleBot Trajectory Generation and Tracking

Shubham Jain Yash Shukla

***Abstract*— We present the implementation of one dimensional and two dimensional adaptive cruise control which follows a given desired trajectory using control lyapunov functions while satisfying constraints specified by a control barrier function to avoid running into obstacles. The Lyapunov function is treated as a soft constraint while the barrier function defines hard constraints for the robot, both of which are satisfied simultaneously using quadratic programming. The controller has been implemented on a turtlebot using Vicon system for feedback.**

I. Introduction

A. Control Lyapunov Barrier Function

Barrier Functions are a type of function whose value on the said point of interest increases to infinity as the point approaches the boundary of a given unsafe region. A barrier function can be considered as a wall that cannot be surpassd. The extension of barrier function to control theory results in Control Barrier Functions (CBF). In dynamical systems, a Lyapunov function helps to determine the stability of the system. A Control Lyapunov Function (CLF) is a lyapunov function for a system with control inputs. That being said, quadratic programming can be used to combine CLF and CBF to achieve the control objectives specified by CLF and the constraint conditions specified by CBF.

B. Adaptive Cruise Control

An interesting application of Control Lyapunov Barrier Functions can be found in autonomous cars. Self driving cars have become a major research topic in the past few years because of their major real-life implications such as increasing safety, comfort and reducing adverse environmental impact of vehicles. Adaptive Cruise Control (ACC), as seen in Fig. 5, is a driver assistive technology that can be used predominantly on highways. It helps the car reach a certain desired velocity and also adjusts the vehicle's speed in response to surrounding vehicles. ACC is an important aspect of self driving cars since it aims at increasing convenience by optimizing the speed of the car to reach the desired goal by minimizing manual effort on part of the driver and also keeping the vehicle safe from collisions. However, this is also a challenging area of research since it incorporates a combination of various objectives which can be in conflict with each other, such as presence of obstacles in the path or presence of traffic not allowing the ego vehicle to reach its desired velocity. Control Lyapunov Barrier Functions (CLBF) can be used to achieve ACC, which requires achieving a desired speed with constraints defined by the safety distance with respect to the vehicles in vicinity.

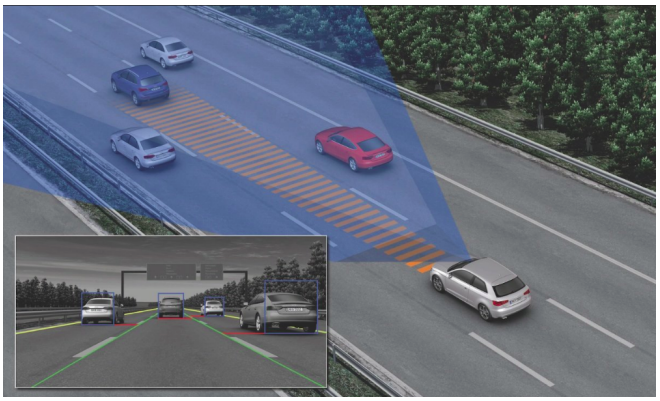

Fig. 1. An artist's rendition of Adaptive Cruise Control showing ego vehicle in white, and surrounding cars in multiple lanes which the ego car also has to consider while trying to maintain its desired velocity

C. Literature Survey

In [1], the control input to the ego vehicle is calculated by combining the control barrier function and the control Lyapunov function through quadratic programming for implementation of adaptive cruise control. This paper assumes a straight line trajectory and only one constraint defined by the barrier function of the vehicle in front. [2] discusses the idea of incorporating multiple Control Barrier Functions which correspond to multiple states of unsafe sets. A Control Barrier Function corresponding to every obstacle has been included which alters the trajectory of the ego vehicle so as to circumvent the obstacle and return to the trajectory for reaching the end goal position. In [3], a combination of Control Barrier Functions and Control Lyapunov Functions or Black-Box Legacy controllers is used through quadratic programming which serves two fundamental purposes. These are adaptive cruise control and lane keeping. Lane Keeping involves correcting the heading of the vehicle to prevent the vehicle from changing the lane thus improving safety of the vehicle. [4] extends the CLF theory to a class of hybrid systems that includes bipedal robots. This paper achieves locomotion for bipedal locomotion via lyapunov functions. [5] introduces exponential CBF to implement high relative degree safety constraints for nonlinear systems. The pole placement helps to smoothly tradeoff between stability of the tracking system and the safety constraints. [6] proposes a control method for combining a stabilzer based on lyapunov functions and safety control based on CBF.

D. Motivation

The Adaptive Cruise Control algorithm explained in [1] had three major limitations.

- The trajectory of both the ego vehicle and front vehicle was restricted to one dimension.
- Front vehicle was constrained to have a constant time invariant velocity, which is rarely the case in real life scenarios.
- Only one vehicle was assumed to be in front of the ego vehicle, thus including only one barrier function, which is that of the front vehicle. However, a real life scenario might contain numerous vehicles or stationary or moving obstacles surrounding the ego vehicle and each of them incorporating a barrier function constraint on the ego vehicle.

In this project, we relax the first two constraints with reference to adaptive cruise control in self driving cars.

E. Problem Statement

The goal of this project is to combine barrier function (hard constraints) and Lyapunov constraint (soft constraint) for the goal of trajectory tracking of a self-driving car in presence of moving cars / obstacles in its surroundings.

As stated in [1], a nonlinear affine control system can be defined by the form:

$$\dot{x} = f(x,z) + g(x,z)u$$
$$\dot{z} = q(x,z)$$

where $x \in X$ are the output states, $z \in Z$ are the uncontrolled states and U are the permissible values for u, the control input.
Quadratic programming combines CLF and CBF into a single controller of the form

$$u^*(x,z) = argmin \frac{1}{2} u^T H(x,z) u + F(x,z)^T u$$
$$u = \begin{bmatrix} u \\ \delta \end{bmatrix} \in R^{m+1}$$

such that $\psi_0(x,z) + \psi_1^T(x,z)u \leq \delta$ is the CLF
and $L_f B(x,z) + L_g B(x,z)u \leq \frac{\gamma}{B(x,z)}$ is the CBF.

Here $H(x,z) \in R^{m+1*m+1}$ and $F(x,z) \in R^{m+1}$ are the arbitrarily chosen cost functions which are based upon the state based weighting of control inputs and L_f is the lie derivative of $V(x,z)$ along the vector field $f(x,z)$ and L_g is the lie derivative of $V(x,z)$ along the vector field $g(x,z)$.

As part of this project, we propose implementing this algorithm on turtlebots in the CIRL laboratory and test its performance for 1 Dimensional path as well as 2 Dimensional trajectories. The desired outcome expected is a smooth adaptation of the ACC on turtlebot by incorporating the real-world limitations such as noisy sensor data, actuator saturation limits and converting the control output to satisfy the required input for the turtlebot.

II. Theoretical Implementation

This section describes the formulation of the control optimization problem using the Control Lyapunov Barrier Functions for the TurtleBot travelling in a straight line (1-D Case) and in a circle (2-D Case).

A. Straight Line path (1-D)

The first step in implementing the ACC involves finding the hard and the soft constraints imposed by the Control Barrier Function and the Control Lyapunov Functions respectively.

The Soft Constraint defined by the Control Lyapnov Function is defined as follows:

$$\psi_0(x,z) + \psi_1(x,z)u \leq \delta_{sc}$$

where,

$$\psi_0(x,z) = -\frac{2(v-v_d)}{m} F_r(x) + \epsilon(v-v_d)^2$$
$$\psi_1(x,z) = \frac{2(v-v_d)}{m}$$

Here, δ_{sc} is the relaxation factor for the soft constraint. Substituting $\delta_s c = 0$ would make the constraint "hard" and would force exact convergence at rate of $epsilon$, v_d is the desired velocity of the ego vehicle and F_r is the aerodynamic drag force as a function of the velocity of the given ego vehicle, v is the current velocity of the ego vehicle, z is the ditance between the ego vehicle and the front vehicle and v_d is the front vehicle velocity

The Barrier Function is defined as follows:

$$B(x) = -log\left(\frac{h(x)}{1+h(x)}\right)$$

where,

$$h(x) = z - 1.8v - \frac{1}{2}\frac{(v_0 - v)^2}{c_d g}$$

Here, c_d is the maximum deceleration possible for the ego vehicle and v_0 is the front vehicle velocity.

The next step involves computing the hard constraint for the Control Barrier Function,

$$\inf_{u \in R}\left[L_f B(x,z) + L_g B(x,z)u - \frac{\gamma}{B(x,z)}\right] \leq 0$$

Where, $L_f B(x,z)$ and $L_g B(x,z)$ are the lie derivatives of $V(x,z)$ along the vector field $f(x,z)$ and $g(x,z)$ respectively and are given by:

$$L_f B(x,z) = -\frac{1.8F_r(x) + m(v_0 - v)}{m(1 - 1.8v + z)(-1.8v + z)}$$
$$L_g B(x,z) = \frac{1.8}{m(1 - 1.8v + z)(-1.8v + z)}$$

The third step involves computing the force based constraints which are given by:

$$u \leq c_a mg$$
$$-u \leq c_d mg$$

The constraints defined above will be combined together by using Quadratic Programming of the form:

$$u^*(x,z) = argmin(\frac{1}{2}u^T H_{acc} u + F_{acc}^T u)$$
$$u = \begin{bmatrix} u \\ \delta_{sc} \end{bmatrix} \in R^{m+1}$$

subject to constraints:

$$\begin{bmatrix} \psi_1(x,z) & -1 \end{bmatrix} u \leq -\psi_0(x,z)$$
$$\begin{bmatrix} L_g B(x,z) & 0 \end{bmatrix} u \leq -L_f B(x,z) + \frac{\gamma}{B(x,z)}$$
$$\begin{bmatrix} L_g B_F(x,z) & 0 \end{bmatrix} u \leq -L_f B_F(x,z) + \frac{1}{B_F(x,z)}$$

where, $H_{acc} = 2\begin{bmatrix} \frac{1}{m^2} & 0 \\ 0 & p_{sc} \end{bmatrix}$ and $F_{acc} = -2\begin{bmatrix} \frac{F_r}{m^2} \\ 0 \end{bmatrix}$ are the cost terms relative to the control and p_{sc} is the penalty term for the soft constraint relaxation δ_{sc}

The controller output is the acceleration of the ego vehicle while the input to the TurtleBot is the linear velocity in that dimension. Conversion from accelration to velocity takes place by the following equation:

$$v_t = v_{t-1} + a * dt$$

Here, v_t is the velocity of the ego vehicle at time t, v_{t-1} is the velocity at time $t-1$ and dt is the value of the time step.

B. Circular trajectory (2-D)

The Kinematic model of an autonomous vehicle can be expressed as:

$$\dot{x} = vcos(\theta)$$
$$\dot{y} = vsin(\theta)$$
$$\dot{\theta} = \omega$$

where v and ω are the input linear and angular velocity respectively, x y are the values of the position in x and y axes and θ is the value of the orientation.

The first step for implementing ACC for a circular trajetory is calculating the tracking error between the reference position and the current position of the ego vehicle.

The tracking error in the vehicle frame is calculated as follows:

$$\begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{ref} - x \\ y_{ref} - y \\ \theta ref - \theta \end{bmatrix}$$

where, $x_{ref}, y_{ref}, \theta_{ref}$ are the values of the reference position and orientation, x, y, θ are the values of the current position and orientation of the ego vehicle and x_e, y_e, θ_e are the errors in the x, y, θ values respectively.

When $x_e, \theta_e \to 0$, y_e becomes uncontrollable. This can be avoided by introducing a change in the variable:

$$\bar{\theta_e} = m\theta_e + \frac{y_e}{\pi_1}$$
$$\text{where, } \pi_1 = \sqrt{x_e^2 + y_e^2 + 1}$$

The system dynamics for the nonlinear affine control system can be defined by the form:

$$\dot{x} = f(x) + g(x)u$$
$$\text{where, } u = \begin{bmatrix} u_0 & u_1 \end{bmatrix}^T \text{ and } x = \begin{bmatrix} \bar{\theta_e} & y_e & x_e \end{bmatrix}^T$$

$$f_x = \begin{bmatrix} \frac{x_e}{\pi_1}\omega_{ref} + \frac{1+x_e^2}{\pi_1^3} v_{ref} sin\left(\frac{\bar{\theta_e}}{m} - \frac{y_e}{m\pi_1}\right) \\ x_e \omega_{ref} + v_{ref} sin\left(\frac{\bar{\theta_e}}{m} - \frac{y_e}{m\pi_1}\right) \\ -\omega_{ref} y_e \end{bmatrix}$$

$$g_x = \begin{bmatrix} m - \frac{x_e}{\pi_1} & \frac{-x_e y_e}{\pi_1^3} \\ -x_e & 0 \\ y_e & 1 \end{bmatrix}$$

The next step involves determining the hard and soft constraints for the control system. The barrier function is defined as:

$$B(x) = -log\left(\frac{h(x)}{1+h(x)}\right)$$
$$\text{where, } h(x) = z - 10$$

In the formulation of $h(x)$, 10 units is the safety distance between the ego vehicle and the front vehicle that the ego vehicle cannot cross. The Hard constraint defined by the Control Barrier Function is defined as:

$$\inf_{u \in R}\left[L_f B(x,z) + L_g B(x,z)u - \frac{\gamma}{B(x,z)}\right] \leq 0$$

Where,

$$L_f B(x,z) = \frac{-v_0 + v_d cos(\theta_e)}{h(x,z)(h(x,z)+1)}$$
$$L_g B(x,z) = \begin{bmatrix} 0 & \frac{1}{10h(x,z)(h(x,z)+1)} \end{bmatrix}$$

The Lyapunov Candidate is given as follows:

$$V = \pi_2 + k_1\pi_1 - (1 + k_1)$$

Where, $\pi_2 = \sqrt{\theta_e^2 + 1}$ and k1 is a constant.
The Lyapunov Soft Constraint is given by the term:

$$\psi_0(x,z) + \psi_1(x,z)u \leq \delta_{sc}$$
$$\text{where, } \psi_0(x,z) = \begin{bmatrix} \frac{\bar{\theta_e}}{\pi_2} & \frac{k_1 y_e}{\pi_1} & \frac{k_1 x_e}{\pi_1} \end{bmatrix} f(x) + \epsilon V$$
$$\text{and } \psi_0(x,z) = \begin{bmatrix} \frac{\bar{\theta_e}}{\pi_2} & \frac{k_1 y_e}{\pi_1} & \frac{k_1 x_e}{\pi_1} \end{bmatrix} g(x)$$

The constraints defined above will be combined together by using Quadratic Programming of the form:

$$u^*(x,z) = argmin(\frac{1}{2}u^T H_{acc} u + F_{acc}^T u)$$

$$u = \begin{bmatrix} u \\ \delta_{sc} \end{bmatrix} \in R^{m+1}$$

subject to constraints:

$$\psi_0(x,z) + \psi_1(x,z)u \leq \delta_{sc}$$

$$\inf_{u \in R} \left[L_f B(x,z) + L_g B(x,z) u - \frac{\gamma}{B(x,z)} \right] \leq 0$$

The output of the controller are the linear velocity v and the angular velocity ω which are then fed to the TurtleBot for trajectory tracking.

III. Practical Implementation

Once we had a working implementation in simulation developed using the theory described above, the next step taken was implementing the same on the turtlebot. The turtlebot is a differential drive robot which can take user-defined velocity and angular velocity values as an input. The aim of our practical implementation was to demonstrate the usability of our ACC algorithms on real systems and dealing with several real-world challenges encountered when dealing with a practical implementation. An overview of our system model is shown in Figure 2. All of the communication was done using ROS framework.

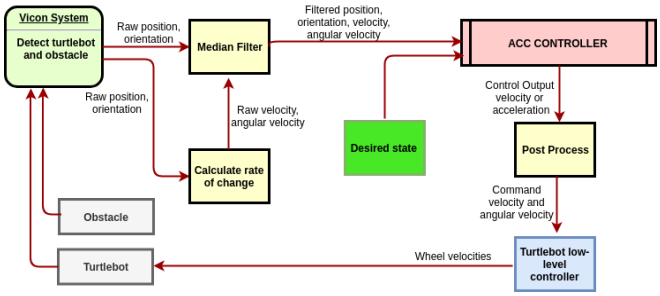

Fig. 2. System model of practical implementation

As seen in Figure 2, there are several important components in our implementation, which are explained in detail in the following sections.

A. Median Filter

On initial testing of the code, we found that raw position and velocity readings obtained from the VICON system are noisy and if used without any filtering, cause the turtlebot to move in a very jerky way and result in collision between the ego vehicle and front vehicle. To rectify this error, we applied a median filter on the position and velocity data for both the bots by using a sliding window of size 10. Fig 3 shows the median filter's performance on a stationary robot. The graph in blue indicate the noisy sensor values which cause the bot to move jerkily. The graph in red is the filter's smoothened response to the noisy values. As we can see, the outliers are being filtered out and the output is much more smoother than the raw input given.

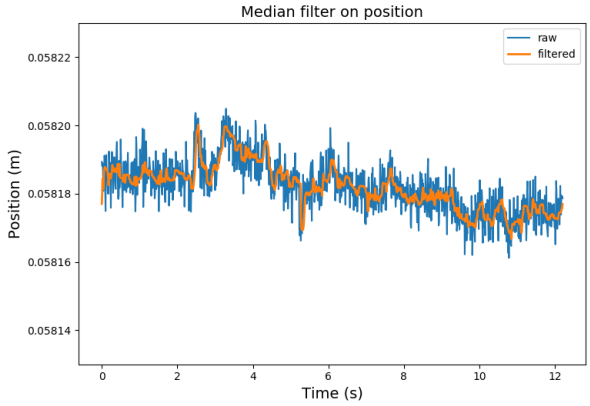

Fig. 3. Median Filter's response to noisy sensor values.

B. Rate change calculation

The Vicon only provides readings of position and orientation of the objects being tracked. However, the Adaptive Cruise Control (ACC) controller required the velocity and angular velocity readings too. These values can be approximated since we know the rate of data capture of our sensor, which is 100 Hz. Thus, the velocity reading at time t is given by $v_t = \frac{x_t - x_{t-1}}{0.01}$ where x_t is the position reading at time t. These raw velocity and angular velocity values are also passed through the median filter described above before being passed on to the controller

C. Desired Trajectory

The desired trajectory of the turtlebot differs in our implementation of 1D and 2D ACC. For the one dimensional case, the turtlebot only has the requirement of attaining a desired velocity of 0.8 m/s. For the 2D case, the turtlebot's desired trajectory is to complete a circle of radius 1 metre in 48 seconds. The desired states of position, velocity, acceleration, orientation and angular velocity are calculated accordingly for each time step by this trajectory generator and passed on to the controller.

The trajectory generator needs an input of certain checkpoints in the trajectory of the form (position, time) and it fits a spline function connecting these checkpoints. The corresponding values of velocity and acceleration are obtained by differentiating the spline function.

D. ACC Controller

This is the actual adaptive cruise control controller which ensures that the turtlebot follows its desired trajectory while maintaining a safe distance from obstacles. The output of the controller in both 1D and 2D cases is different and is handled appropriately by the post processing stage.

- 1D case - The controller output is an acceleration command for the turtlebot.

- 2D case - The controller output is a velocity and angular velocity command for the turtlebot.

E. Post Processing

Since the output of controller is different, the post processing steps differ for the 1D and 2D case. The output of both cases, however, is a velocity and angular velocity vector which is passed on to the low level controller of the turtlebot for conversion into wheel velocities.

- 1D case - For the 1D case, since the controller output is acceleration, we encounter the problem of actuator saturation. The turtlebot has a lower saturation value of 0.04 m/s which means that given a velocity command lower than this value, it remains at rest. This causes problems if, for example, the turtlebot starts from rest. In such a case, if the new velocity calculated using the acceleration output of controller is less than the saturation limit, then the turtlebot continues to remain at rest. To counter this problem, an internal velocity state is maintained for the turtlebot which is updated at every time step using the acceleration command obtained from the controller. The value of this internal velocity is then used as the command velocity for the turtlebot. To ensure there is no accumulation of error, when the difference between the internal velocity and actual velocity of the turtlebot exceeds a certain threshold, the value of internal velocity is reset to the actual velocity of turtlebot. Also, since this is a 1D case, the angular velocity command given is always zero.

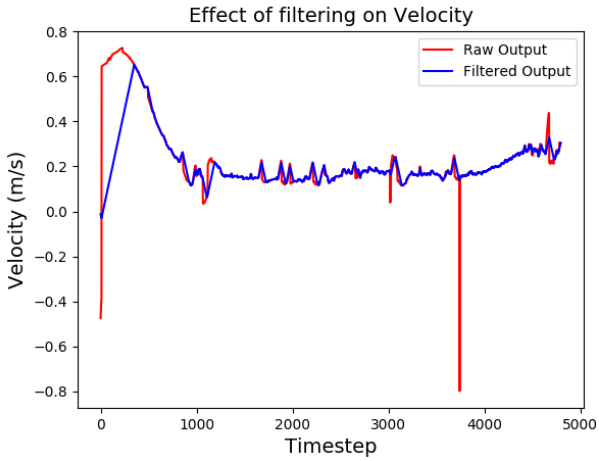

Fig. 4. Velocity filter applied to controller output

- 2D case - For this case, the controller output is velocity and angular velocity, which, in theory can be passed on directly to the low level controller of the turtlebot. But, as we saw before, the inputs to the controller are noisy due to uncertainties in the Vicon readings. Hence, the output of the controller is also noisy with sudden spikes. This leads to jerky motion of the turtlebot if controller output is directly used. We get rid of this jerky motion by smoothening the velocity output which is achieved by remembering the velocity command given to the turtlebot in the previous timestep(v_{t-1}). If the difference of velocity output v_t and v_{t-1} is greater than a predefined threshold T, then the current velocity command is reduced to $v_t = v_{t-1} + T$. For our case, a threshold of 0.002 m/s was used because it corresponds to an acceleration on $0.2m/s^2$ for a controller frequency of 100 Hz. The effect of velocity filtering can be seen in Figure 4 where we see that sudden spikes in velocity are smoothened out with much more gradual changes in velocity.

IV. Results

In this section, we present the results of our implementation of adaptive cruise control on the turtlebot, for both 1D and 2D cases.

A. 1-Dimensional Adaptive Cruise Control

Fig 5 indicates the position response of both the vehicles with respect to time. As observed, the ego vehicle constantly maintains a safe distance of 0.5 meters given by the hard constraint defined by the CBF.

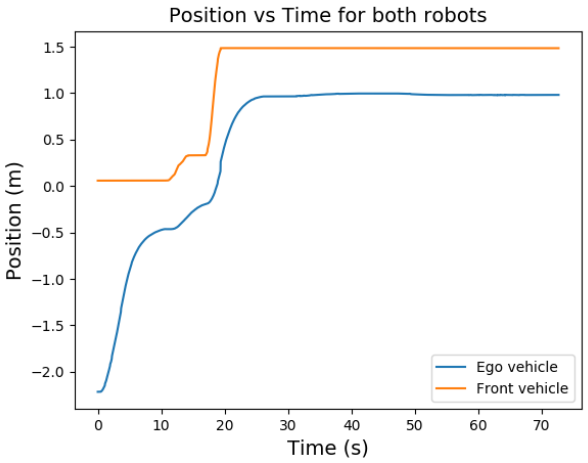

Fig. 5. Distance between the vehicles as a function of time.

Fig 6 indicates the velocity response of both the vehicles with respect to time. A phase shift between the velocities of the ego vehicle and the front vehicle is observed. A similar phase shift was observed in the simulated sinusoidal velocity profiles as shown in Figure 7. This is because the boundary distance of the barrier function changes with velocity and hence the ego car has to adjust accordingly. Hence, we see that the real robot implementation behaves quite similar to the simulation.

Here is a link to the video of our ACC implementation in 1-Dimension on the turtlebot in CIRL laboratory.

B. 2-Dimensional Adaptive Cruise Control

For this case, the turtlebot's desired trajectory is a circle of radius 1 m. We introduce obstacles intermittently in its path when it is following its desired trajectory to test its response

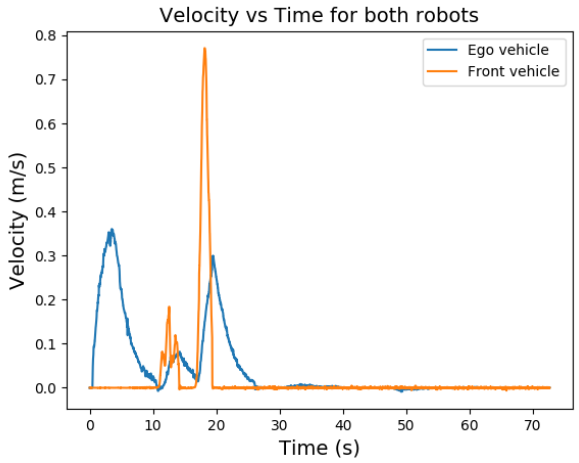

Fig. 6. Velocity of turtlebot and obstacle as a function of time.

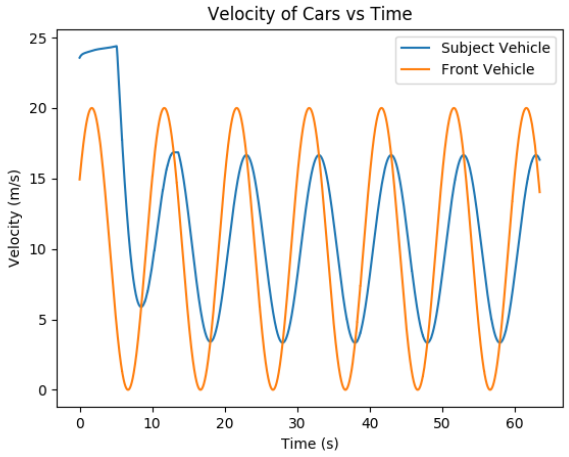

Fig. 7. Velocity of cars as a function of time in simulation.

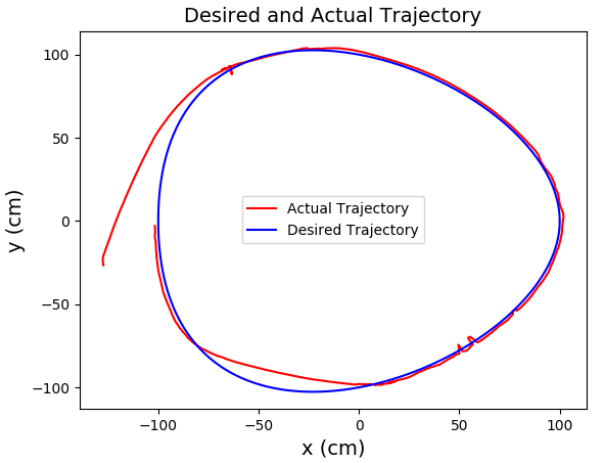

Fig. 8. Plot showing desired trajectory and actual trajectory of turtlebot inspite of presence of obstacles in its path

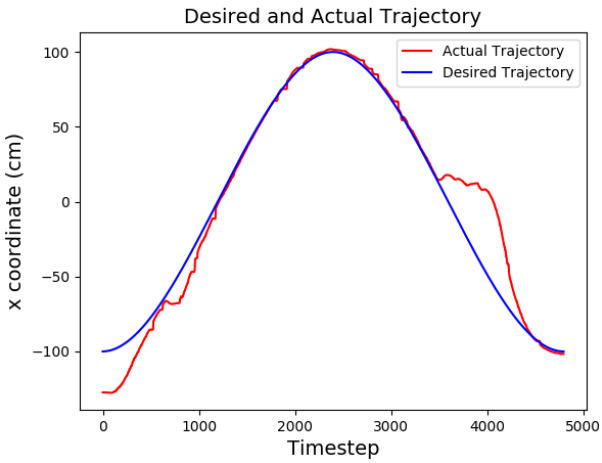

Fig. 9. Plot showing desired and actual x coordinate of turtlebot w.r.t time. We can see that the turtlebot deviates from its desired path when an obstacle is placed in its path but again reaches its desired trajectory once the obstacle is removed

to barriers in its path. It was observed that the controller has an offset of 0.4 units from the desired trajectory. Hence, to reduce the error, the units used for the controller was in centimetres so that error is 0.4 cm. We tried using millimetres too for further reducing error but it ended up making the controller unstable. The response of controller with obstacles in its path is shown in Figure 8 and 9

Here is a link to the video of our ACC implementation in 2-Dimension on the turtlebot in CIRL laboratory.

V. Conclusions

We have successfully demonstrated a working implementation of 1D (video) and 2D (video) adaptive cruise control. Several real-world problems were encountered in the practical implementation which were dealt with accordingly. The entire communication model was based on a ROS based framework. We obtained codes for simulation of ACC at the start of our project but re-implemented 2D adaptive cruise control in Python so that it is compatible with ROS and can run easily on the turtlebot. Some bugs in the existing simulation code were fixed and code was made more modular. The results demonstrate that the response of controller in real world is quite similar to simulation for both 1D and 2D case.

VI. Future Scope of Work

Although results for our initial implementation of ACC on the turtlebot are quite satisfactory, there is a lot of scope for improvement in terms of smoothness of controller and it's expansion to more general cases. Some possible future steps are mentioned below -

- The control output for the angular velocity for the 2-Dimensional trajectory is irregular sometimes which results in shaky motion of the robot. Steps can be taken in this direction to smoothen it out.
- It was observed that the ego vehicle tried to exactly satisfy its hard constraint, that is, the control barrier

function. However, this is not possible because of noise and lag in the sensor values, which causes the robot to oscillate very slowly at the boundary of the barrier function. This can be improved by providing a buffer zone of values for the ego vehicle to come to rest.

- In some cases it was observed that the turtlebot was not always successful in finding a way to circumvent around the obstacle. Hence, more smart ways of trajectory recalculation can be implemented to ensure that the robot does not get stuck.
- The existing formulation for barrier functions can be generalised for handling multiple obstacles as well by using weighted barrier functions.

REFERENCES

[1] A. D. Ames, J. W. Grizzle, P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control", Proc. IEEE 53rd Annu. Conf. Decis. Control, pp. 6271-6278, Dec. 2014.

[2] M. Z. Romdlony, B. Jayawardhana, "Stabilization with guaranteed safety using control Lyapunovbarrier function", Automatica, vol. 66, pp. 39-47, 2016.

[3] X. Xu, J. W. Grizzle, P. Tabuada, A. D. Ames, "Correctness guarantees for the composition of lane keeping and adaptive cruise contro", 2016, [online] Available: http://arxiv.org/abs/1609.06807.

[4] A.D. Ames, K. Galloway, K. Sreenath, J.W. Grizzle, "Rapidly exponentially stabilizing control lyapunov functions and hybrid zero dynamics", Automatic Control IEEE Transactions on, vol. 59, no. 4, pp. 876-891, April 2014.

[5] Q. Nguyen, K. Sreenath, "Exponential control barrier functions for enforcing high relative-degree safety-critical constraints", Proc. American Control Conf., pp. 322-328, 2016.

[6] Romdlony, M.Z. and Jayawardhana, B. (2014). Uniting control Lyapunov and control barrier functions. In IEEE Conference on Decision and Control, 22932298.