Real-time Safe Control for Uncertain Goal State

By Zachary Block, Amar Mohanty, and Harshil Parekh

Abstract– Low quality sensors introduce significant noise to its measurements which can be problematic for safe control when trying to follow a target moving non-linearly. The goal of this project is to develop an autonomous system that can plan in real-time a safe trajectory to arrive to and follow a moving target when there is uncertainty in the target's state. With safety in mind, we propose a novel approach of estimating this uncertainty using a particle filter to choose a point closest to the mean as the target. Additionally, we have developed an online trajectory planner that fits a quintic spline along waypoints which utilizes a hybrid bang-bang controller with MPC. This approach resulted in no collisions with a moving target and was able to follow the target better than traditional techniques.

I. Introduction

Robotic applications make use of sensors ranging from camera systems to LIDARs. Measurements from these sensors are always accompanied with some noise. Based on the quality of the sensor the range of noise can vary. This can be problematic in the case of low quality sensors when tracking a moving target because if we are not confident where the target is then we are likely to collide with or miss the target. This is a challenging problem because the target dynamics and the noise associated with the measurements are unknown. Additionally, it is difficult to plan trajectories online while the target state is being estimated for a non-linear motion which is computationally expensive.

Thus, the goal of this project is to develop an autonomous system that can plan in real-time a safe trajectory to arrive to and follow a moving target when there in uncertainty in the target's state. Safe control is defined as a collision free trajectory with a short time of arrival to the target, maximized for time following the target while being close to it's true position. Consequentially, we propose a novel approach of estimating the uncertainty using a particle filter and choosing the point closest to the mean as the target. Additionally, we developed an online trajectory planner that fits a quintic spline along waypoints which utilizes a bang-bang controller with MPC. For this experiment, we have three independent systems: the target is a tractor moving non-linearly in an open field, the system is an autonomous cart following the tractor, and the sensor is a noisy camera system on a small drone relaying the state of the target. Using just the sensor data, the autonomous trailer must reach the tractor in a minimum amount of time, all without any risk of collision

II. Background

Since we have a decoupled camera system relaying the target state it is safe to assume there can be multiple reasons for the readings to have random noise. In order to counter this we implement a Real-time Particle Filter[1] to provide an accurate estimate of the target position. Initially the project was aimed at countering the uncertainty in goal state for a Linear Parameter Varying Dynamic Systems (LPVDS)[2]. The LPVDS approach introduced a learning of the system dynamics component to fit a Gaussian mixture model over the non-linear trajectory. The system would then select a Gaussian based on the robot state which would provide stable linearized dynamics. For better analysis of the advantages of the implemented approach we pivoted to simpler dynamics elaborated in the methodology section.

III. Methodology

A. System Dynamics

The robot is an autonomous cart whose state has a x and y position value and a heading measured by theta as shown in the figure below. Below is the system dynamics equation.

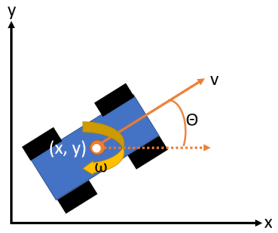

Fig. 1. Diagram of the robot cart

Note that the robot has minimum and maximum velocity and acceleration constraints.

$$q_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \cos(\theta_{t-1})dt & 0 \\ \sin(\theta_{t-1})dt & 0 \\ 0 & dt \end{bmatrix} \begin{bmatrix} v_{t-1} \\ \omega_{t-1} \end{bmatrix} \tag{1}$$

The flat outputs of the system can be derived from the system dynamics equation.

$$\theta_{t-1} = \tan^{-1}\left(\frac{y_t - y_{t-1}}{x_t - x_{t-1}}\right) \tag{2}$$

$$v_{t-1} = \frac{x_t - x_{t-1}}{\cos(\theta_{t-1})dt} \tag{3}$$

$$\omega_{t-1} = \frac{\theta_t - \theta_{t-1}}{dt} \tag{4}$$

B. Sensing

The particle filter is a recursive Bayesian estimation algorithm that is used to estimate the state of a target based on noisy measurements from sensors. It is particularly useful for estimating the state of a non-linear target. The output of the filter is a probability distribution of the target's state.

The filter works by sampling the space of possible target states with initial guesses, called particles. As new measurements

are received, the particles are updated and their weights are adjusted based on the likelihood of the measurement given the particle's state. The output of the filter is a probability distribution of the particles, which can be used to calculate the weighted mean and covariance of the distribution. The filter operates in a loop, updating the particles and their weights based on new measurements as they are received.

Once we calculate the heading and distance of the new target from the old target we pass it to the *predict* function (5). This function uses the estimated dynamics u_k of the target to then propagate each particle $i = 1, ..., n$ forward by adding the same distance along x and y position using the heading calculated. Here f is the system dynamics using noise ϵ_k. Weights of the particles are unchanged $w^{(i)}_{k+1|k} = w^{(i)}_{k|k} = \frac{1}{n}$.

$$x^{(i)}_{k+1|k} = f(x^{(i)}_{k|k}, u_k) + \epsilon_k \tag{5}$$

The particle filter also receives as measurement, a relative position of the target with respect to landmarks such as walls, doors or objects in order to localize. The next step is to incorporate this observation once the particles have been propagated. Using this new observation y_{k+1}, we update the weight of each particle using the likelihood of receiving that observation in the update function (6). The weights are then normalized to always add up to one. Note that $P(y_{k+1}|x^{(i)}_{k+1|k})$ is a Gaussian depending on some observation noise which is simulated.

$$w^{(i)}_{k+1|k+1} \propto P(y_{k+1}|x^{(i)}_{k+1|k}) w^{(i)}_{k+1|k} \tag{6}$$

Finally we perform a structured importance sampling step to obtain new particle locations $x^{(i)}_{k+1|k+1}$ with the updated uniform weights $w^{(i)}_{k+1|k+1} = \frac{1}{n}$. This is done by the *systematic resample* and *resample from index* functions. The output of which is a denser distribution with higher probability of having the target than the one the filter started out with. This probability distribution is analyzed by calculating its weighted mean and covariance matrix using the *estimate* function, following which the *omega star* function in the robot class determines the point in the distribution the robot must reach. In order to test different scenarios for probability of collision we pick ω^* to be any of the four points the mean of the distribution represented as an orange dot in figure 2, the particle with maximum weight represented as a red 'x', the point on minimum axis of the approximated ellipse and the point on the maximum axis of the ellipse closest to the robot.

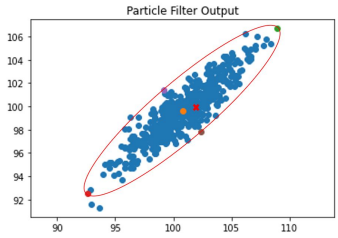

Fig. 2. Probability distribution output of the particle filter with mean at (100.78,99.62) approximated as an ellipse

C. Trajectory Generation & Control

As stated previously, since the system can be written in terms of its flat outputs, control of the system can be achieved via the creation of a continuous trajectory of both the x and y coordinates of the car. For the purposes of this project, trajectory generation was broken into three parts: Waypoint Generation, Trajectory Constraint Enforcement, and Trajectory Optimization.

1) Waypoint Generation: For most cases, the waypoints used for spline creation will be the robot's current state (position, velocity, and acceleration) and the ω^* state. However, this gives rise to a possible collision event if the robot is in the area shown below:

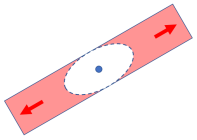

Fig. 3. The areas along the axes of maximum uncertainty are known as the "No-Fly-Zone" due to the increased likelyhood of collision. Linear trajectories generated from inside this region to the axis of minimum uncertainty will result in said trajectory crossing into the uncertainty region.

As such, the trajectory generation algorithm (TGA) will generate a protrusion waypoint, which will force the robot outside the "No-Fly-Zone" before approaching ω^*. The position of this new waypoint P is calculated using the equations below:

$$\theta_{\omega^*} = arctan2(\omega^*_Y - T_Y, \omega^*_X - T_X) \tag{7}$$

$$P_X = \frac{R_X + T_X}{2} + cos(\theta_{\omega^*}) \sqrt{(\omega^*_X - R_X)^2 + (\omega^*_Y - R_Y)^2} \tag{8}$$

$$P_Y = \frac{R_Y + T_Y}{2} + sin(\theta_{\omega^*}) \sqrt{(\omega^*_X - R_X)^2 + (\omega^*_Y - R_Y)^2} \tag{9}$$

Where T refers to the target state, and R refers to the robot state. As shown below, this additional waypoint pushes the trajectory away from the uncertainty region.

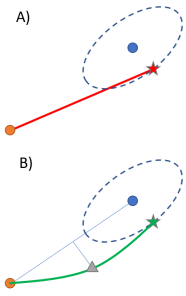

Fig. 4. The trajectory in A shows how the robot could enter the uncertainty region with only two waypoints. The trajectory in B was calculated using the additional protrusion waypoint, which ensures the robot does not enter the uncertainty region

2) Trajectory Constraint Enforcement: During the splining process, it is possible to constrain the trajectory at each waypoint. These constraints can be equality or inequality constraints. For instance, setting a velocity equality constraint on the final waypoint ensures that the trajectory ends with that specified velocity. For cases where inequality constraints are used, the trajectory will be constrained to values above or below the specified value at that waypoint only. In the case of the toy car, which has kinematic constraints such as maximum velocities, accelerations, and jerk values, performing a regular spline is not enough to ensure the trajectory remains below these max values between waypoints. Normally, these trajectory constraints can be implemented via methods like iLQR or multiple shooting at the cost of execution time. Since the goal of this project is to develop safe trajectories in real time, Daniel Mellinger's and Vijay Kumar's concept of temporal scaling was used.

Essentially, the time needed to traverse a trajectory between two waypoints can be scaled in time, and the generated trajectory will reflect this change with subsequent changes to velocity, acceleration, jerk, etc. This algorithm aims to calculate a close to optimal time for each segment such that the kinematic constraints are not violated throughout the entire trajectory. This is done by calculating the trajectory breakpoints, or points along the trajectory where the specified maximum velocity, acceleration, etc. are reached, then back calculating a total trajectory time based off these positions. One caveat to this method is that trajectory time is calculated based on the assumption that velocity, acceleration, and jerk are zero at the beginning and end of the trajectory. While the resulting time is not optimal in cases where the trajectory is recalculated while the robot is in already in motion, this method is incredibly quick to execute, and it guarantees the robot will never exceed its kinematic constraints.

Since the generated splines are quintic, the system is considered fifth order and only the constraints on the position, max velocity, and max acceleration are considered. The algorithm begins by calculating the acceleration breakpoint (A_{BP}), or the final position of a trajectory during which the acceleration reaches its maximum allowable value.

$$A_{BP} = \frac{2A_{max}^3}{J_{max}^2} \tag{10}$$

Then, the velocity breakpoint (V_{BP}), or the final position of a trajectory during which the velocity reaches its maximum allowable value, is calculated via the following equation:

$$V_{BP} = \frac{J_{max}V_{max}^2 + V_{max}A_{max}^2}{J_{max}A_{max}} \tag{11}$$

The distance the robot needs to travel along this axis (X or Y) is then compared to these two values to determine whether the trajectory will have seven, five, or three segments. If the travel distance (D) is larger than V_{BP}, a seven segment trajectory time is used. If the travel distance is larger than A_{BP} but smaller than V_{BP}, a five segment trajectory time is used. If the travel distance is smaller than both breakpoints, a three segment trajectory time is used.
These trajectory times (T_7, T_5, and T_3 respectively) are calculated using the equations below:

$$T_7 = \frac{A_{max}}{J_{max}} + \frac{V_{max}}{A_{max}} + \frac{D}{V_{max}} \tag{12}$$

$$T_5 = \frac{A_{max}}{J_{max}} + \frac{\sqrt{4A_{max}D + \frac{A_{max}^4}{J_{max}^2}}}{A_{max}} \tag{13}$$

$$T_3 = \sqrt[3]{\frac{32D}{J_{max}}} \tag{14}$$

Once calculated, the trajectory time is then used to generate an appropriate quintic trajectory spline.

3) Trajectory Splines: Using the waypoints and trajectory time detailed in the previous section, the algorithm can now generate the quintic splines representing the robot's trajectory. Since there will be one fifth order equation for each of the flat outputs, there needs to be 12 total constraints for each spline. For two waypoint trajectories, these constraints come in the form of kinematic equality constraints (position, velocity, and acceleration) at both the start and end waypoints. For three waypoint trajectories, there should be a total of 24 constraints, 12 for each spline. Each waypoint will have position equality constraints, with only the end points have velocity and acceleration equality constraints. The middle waypoint has continuity constraints on velocity, acceleration, jerk, and snap. These constraints, along with the acceleration cost, allow Drake to optimize a quadratic programming problem, and determine the optimal trajectory.

4) Model Predictive Control: In addition to the above trajectory generation efforts, our group also implemented a rudimentary model predictive control loop. Essentially, the robot is able to keep track of the last N target positions, where N is a tunable hyperparameter, which allows the robot to calculate the target's average velocity, acceleration, and jerk. Using these values, the robot then runs a kinematic analysis, predicting the future position, velocity, and acceleration of the target using kinematic equations of motion.

IV. Results and Analysis

We evaluated our trajectory and noise algorithm to determine the optimal hyper-parameters for safe control. The robot is considered to have safe control if it had 0 collisions with the target, was able to arrive to w* quickly, followed w* consistently, and was near the target. Thus, we measured the number of collisions the robot had with the target, time it took to arrive to the target. Additionally, once the robot arrived to w*, we measured what percentage of time the robot followed w* and the average distance away the robot was from the true target location. We tested this with a variety of target trajectories (static, linear, triangle, and sine) to test the robustness of our trajectory algorithm and tune our hyper-parameters as shown in the figure below[1].

[1]Link to the animated gifs of the target trajectories.

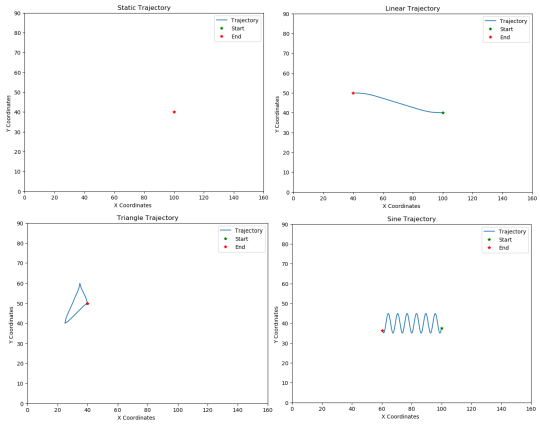

Fig. 5. Static, linear, triangle, and sine trajectory of the target.

We evaluated each of the W* star methods with all four trajectories and averaged all the measured values as shown in the table below. We found that the at the cost of being farther away from the true target, we were able to have no collisions with the target when using the min or max axis methods. Additionally, the robot was able to follow w* significantly better utilizing the min and max axis methods than the mean and max weight methods. One theory behind the improvement with following is that the mean and max weight values may be jumping around more as we get new sensor readings making it harder for the robot to follow than the min or max axis methods[2].

Varying W* for All Trajectories				
	Mean	Max Weight	Min Axis	Max Axis
Collisions (samples)	1250.88	1122.75	0.00	0.00
Time of Arrival (s)	20.93	28.71	18.03	22.89
% Followed	32.02	34.55	60.92	60.67
Avg Distance to Target (px)	2.41	1.81	3.58	5.72

Fig. 6. W* method impact on safety performance for all target trajectories.

When we remove the static trajectory case, the performance of the mean and max weight methods reduces significantly more than the min and max axis methods as shown in the table below. In other words, the min and max axis methods perform better with moving targets than traditional approaches like using the mean or max weight.Thus, the min axis method is the safest approach w* method because it had no collisions, followed w* consistently, and had a shorter average distance than the max axis.

Varying W* for Moving Trajectories				
	Mean	Max Weight	Min Axis	Max Axis
Collisions (samples)	402.33	629.67	0.00	0.00
Time of Arrival (s)	20.59	27.02	16.97	21.22
% Followed	9.56	13.18	48.76	48.72
Avg Distance to Target (px)	3.13	2.34	3.52	5.69

Fig. 7. W* method impact on safety performance for moving target trajectories.

[2]Link to the animated gifs of the w* methods.

Next, we varied the number of particles used in the particle filter to find the smallest particle number that performs well since the particle number is proportional to computational time. Varying the particle number, affected the % followed the most. Most likely, the reduction in particle number, led to particle estimates jumping more. We found that 200 particles was the best balance between performance and computational time as shown in the table below.

Particle Number Affect on Performance of Particle Filter				
	100 Particles	200 Particles	300 Particles	400 Particles
Collisions (samples)	0.00	0.00	0.00	0.00
Time of Arrival (s)	18.37	18.13	18.01	18.04
% Followed	55.36	59.79	60.67	61.17
Avg Distance to Target (px)	3.55	3.57	3.59	3.57

Fig. 8. Particle number impact on safety performance.

Finally, we found that varying the MPC window size affected how well the robot could follow the target as shown in the table below. Without any MPC control, the robot couldn't follow the target as well since there was no prediction of where the target would go being accounted for in the trajectory planning. Conversely, making the window size too large also reduced the robot ability to follow the target. Thus, the MPC window size for 10 steps had the best performance since it had the highest % followed and shortest time of arrival.

Varying the MPC Window Size				
	None	10	20	50
Collisions (samples)	0	0	0	0
Time of Arrival (s)	16.435	16.46	16.52	16.935
% Followed	44.625	46.27	45.45	40.275
Avg Distance to Target (px)	3.78	3.67	3.62	3.695

Fig. 9. MPC window size impact on safety performance.

V. Conclusion

Overall, the trajectory generation algorithm paired with the particle filter was able to both accurately determine the target's position based on noisy input data and also generate safe, collision-free trajectories. By tuning certain hyperparameters, the performance of our robot was able to be optimized, allowing for better following accuracy, quicker arrival times, and less likelyhood of collision. As discussed previously, setting ω^* to either the minimum or maximum axis of uncertainty eliminates the possibility of collision in our experimentation. Additionally, setting the particle count parameter to 400 improves following percentage with a negligible increase in arrival time.

While there are several areas which can be improved: the speed of the algorithm, hyperparameter tuning, and MPC robustness, these experiments show that this approach is a viable option for this type of problem. With more work on these areas, this approach could provide a quick and computationally cheap way to perform on the fly trajectory generation for moving targets.

VI. Acknowledgement

We would like to thank the University of Pennsylvania for providing us the opportunity to explore this project as a part of the curriculum for course MEAM5170: Control and Optimization With Applications in Robotics taught by Dr.Michael Posa.

References

1) Kwok, Cody & Fox, Dieter & Meilâ, Marina. (2004). Real-Time Particle Filters. Proceedings of the IEEE. 92. 469 - 484. 10.1109/JPROC.2003.823144.
2) Y. Shavit, N. Figueroa, S. S. M. Salehian and A. Billard, "Learning Augmented Joint-Space Task-Oriented Dynamical Systems: A Linear Parameter Varying and Synergetic Control Approach," in IEEE Robotics and Automation Letters, vol. 3, no. 3, pp. 2718-2725, July 2018, doi: 10.1109/LRA.2018.2833497.
3) Mellinger, D., & Kumar, V. (2011). Minimum snap trajectory generation and control for quadrotors. Paper presented at the 2520-2525. https://doi.org/10.1109/ICRA.2011.5980409
4) García-Martínez, J. R., Rodríguez-Reséndiz, J., & Cruz-Miguel, E. E. (2019). A new seven-segment profile algorithm for an open source architecture in a hybrid electronic platform. Electronics (Basel), 8(6), 652. https://doi.org/10.3390/electronics8060652