Motion Planning with Dynamic Constraints Through Pose Graph Optimization

by

Nadya L. Balabanska

B.S. Computer Science
Massachusetts Institute of Technology, 2019

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2020

Signature of Author: _____
Department of Electrical Engineering and Computer Science
August 14, 2020

Certified by: _____
Sertac Karaman
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by: _____
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Motion Planning with Dynamic Constraints Through Pose Graph Optimization

by

Nadya L. Balabanska

Submitted to the Department of Electrical Engineering and Computer Science
on August 14, 2020 in Partial Fulfillment of the Requirements for
the Degree of Master of Engineering in Electrical Engineering and
Computer Science

ABSTRACT

This contribution is an optimization-based method for robotic path-planning that is able to recover vehicle controls in addition to discovering an optimized, feasible trajectory from start to goal for vehicles with arbitrary dynamics. The motion planner extends the application of factor-graph optimization commonly used in simultaneous localization and mapping tasks to the path-planning task, specifically the "timed elastic band" trajectory optimization approach [1] for control input extraction functionality. This is achieved by the introduction of control input-dependent vertices into the factor-graph along with a way to systematically design dynamics violation costs without relying on hand-picked geometric parameters. An implementation of the planner successfully recovers vehicle control inputs and produces feasible trajectories in simulation testing.

Thesis Supervisor: Sertac Karaman
Title: Associate Professor of Aeronautics and Astronautics

Contents

List of Figures

1 Introduction and Background

The motion planning problem in robotics refers to finding a sequence of valid configurations for a mobile robot starting at some initial state and trying to arrive at a specified goal state. Robot motion operates in a *configuration space C* [1], which is the vector space of all possible configurations, or poses of the robot. Planning problems are defined in terms of a *state space*, which describes all the possible situations which could arise. A *state* could, for example, represent a position and orientation of the robot [2].

Low-dimensional problems can be solved with grid-based algorithms that overlay a grid on top of the configuration space, or geometric algorithms that compute the shape and connectivity of the free space. For high-dimensional systems under complex constraints, exact motion planning is computationally intractable. Potential-field algorithms, which treat the robot as a particle under the influence of an artificial potential field constructed to locally reflect the structure of the free C-space, are efficient, but fall prey to local minima [2].

Sampling-based path planning algorithms, on the other hand, conduct a search that probes the C-space instead of constructing the obstacle space directly. A sampling-based algorithm samples the C-space and uses a collision detection module to handle the particular geometric models. Sampling-based planners are currently considered state-of-the-art for motion planning in high-dimensional spaces, and have been applied to problems which have dozens or even hundreds of dimensions. Multi-query planners build a roadmap of the entire environment that can be used for multiple queries. The Probabilistic Roadmap algorithm (PRM) [3] is an example of a multi-query planner. The algorithm builds up a roadmap across the space by taking valid random samples from the space and connecting the configurations to each other. Once a start and goal configuration are added, a graph search algorithm is employed to find a path within the roadmap. Single-query planners planners typically grow a tree of states connected by valid motions. An example of a single-query planner is the Rapidly-exploring Random Tree (RRT) algorithm [4], which incrementally builds a tree of states sampled randomly from the search space beginning at the start state and is biased to grow towards large unsearched areas of the space.

In some cases, finding any valid path between the start and goal states is not enough; the problem may require optimizing some path quality metric. This metric could be the length of the path, or it could be some general cost framework inclusive of a variety of path properties such as path length, clearance from

obstacles, and any metric specific to the problem and robot. Optimizing variants of standard path-planning algorithms exist. PRM* [5] and RRT* [6] are respectively asymptotically optimal variants of the PRM and RRT algorithms described above.

Another optimization method is trajectory modification, which modifies an initial feasible trajectory that a global planner generated beforehand. Dynamic modification of a preplanned path is beneficial over offline trajectory planning for situations that require coping with changes of a dynamic environment by incorporating the most recent sensor data for local refinement of the trajectory [2]. In most realistic applications the model of the environment is subject to continuous change due to partial, incomplete maps and dynamic obstacles, and the computation of a large scale global path is often not feasible in real-time applications. Thus local modifications to the trajectory are beneficial. An example of a trajectory modification algorithm, the CHOMP algorithm, improves the quality of sampled trajectories using covariant gradient techniques [7].

Additionally, robots have *differential constraints*, which are the restrictions on allowable velocities for the vehicle at each point. These constraints may for example arise out of the kinematics and dynamic constraints of the robot [8]. Many robot motion planning applications ignore dynamics and other differential constraints and focus primarily on the translations and rotations required to move the robot. Many path-planning solutions apply such path-planning techniques as explained above without considering dynamics constraints, and then use control techniques to ensure that a computed path is executed as closely as possible. For example, the planned path could be the result of running a sampling-based approach (without considering dynamics) and a pure pursuit controller [9] can be used to follow the path. This type of solution relies on the assumption that a path can be easily determined between any two configurations in the absence of obstacles. For example, the sampling-based roadmap approach assumes that two nearby configurations can be connected by a "straight line" in the configuration space [2].

However, this is not always a reasonable assumption. Some vehicles have dynamics constraints that require consideration in the path-planning stage, because a path generated without considering the robot's dynamics would not be meaningful. An example of such a system would be a vehicle gliding on the an icy surface. In cases like these, it is beneficial to consider the vehicle's dynamics constraints in the planning process. Incorporating the vehicle dynamics in the path-planning stage yields trajectories which are perfectly feasible for the vehicle to follow.

Control theory refers to designing inputs to physical systems described by differential equations. In control theory literature, motion planning sometimes refers to the construction of inputs to a nonlinear dynamical system that drives it from an initial state to a specified goal state. This is the problem targeted in this contribution. Rather than using geometric models as in the aforementioned approaches, models based on differential constraints will be considered [2].

Differential models are typically expressed as $\dot{x} = f(x,u)$. The problem becomes finding a sequence of control inputs $u(t,x)$ such that the robot will travel from the start state x_{start} to the goal x_{goal}.

Extensions to path-planning algorithms can be made to incorporate dynamics constraints. RRT can be easily modified to incorporate dynamics [10]. States are sampled as mentioned previously and the inverse value problem is solved to obtain the controls to arrive at the state at each extension of the tree, for example. This can slow down planning time though, since the inverse value problem can be computationally expensive, and is run at each sample step. Trajectory modification may be a more efficient approach in this case [2]. A trajectory modification planner with dynamics is the timed elastic band method, which is expanded on in this contribution.

The target contribution is a flexible motion planner that

1. handles simple specifications of arbitrary dynamics models
2. provides a simple and flexible way to define an optimization objective for the trajectory
3. optimizes both trajectory and control inputs based on the objective with reasonable efficiency

1.1 Timed-Elastic Band

The Timed-Elastic Band [11]approach is a trajectory modification approach. The Timed Elastic Band (TEB) as a motion-planning approach optimizes trajectories by modifying an initial trajectory generated by a global planner and formulating the motion-planning problem as a scalarized multi-objective optimization problem.

The Timed-Elastic Band represents the problem of trajectory optimization in a hypergraph structure. The trajectory is described in terms of a sequence of robot poses and the time intervals between two consecutive

poses. The TEB is defined as a tuple of both these sequences:

$$B := (Q, \tau) \tag{1}$$

$$Q = \{q_i\}_{i=0...n} \tag{2}$$

$$\tau = \{\Delta T_i\}_{i=0...n-1} \tag{3}$$

where q_i corresponds to the vehicle pose with index i along the trajectory, and ΔT_i corresponds to the time difference between consecutive poses q_i and q_{i+1} in the trajectory.

The hypergraph is optimized using the weighted sum model:

$$f(B) = \sum_k \gamma_k f_k(B) \tag{4}$$

$$B^* = \text{argmin}_B f(B) \tag{5}$$

where B^* denotes the optimized TEB, $f(B)$ is the underlying global objective function, formulated in terms of cost functions that penalize the violation of a constraint, and γ_k is a variable multiplier for each error function allowing harsher penalization for some constraints violations than others [11].

Because most objectives are local, relating only to parameters associated with few consecutive robot states, the resulting system matrix is sparse, allowing the use of fast and efficient optimization techniques. The formulation of the problem into this type of hypergraph representation serves as the inspiration for this contribution.

1.1.1 Hyper-graph Optimization

Optimizing the factor graph requires solving a sparse scalarized multi-objective optimization problem. An additional advantage to this approach is the ease of modularizing objectives and constraints.

The nonlinear optimization problem has the following form

$$\mathbf{F}(x) = \sum_{k=i,j\in C} \mathbf{F}_k = \sum_{k=i,j\in C} \mathbf{e}_k(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})^T \Omega_k \mathbf{e}_k(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij}) \tag{6}$$

$$\mathbf{x}^* = \min_x \mathbf{F}(x) \tag{7}$$

where $\mathbf{x}$ is the parameter to be optimized, which is replaced with the TEB tuple B. By means of using scalar error terms, $\mathbf{F}_k$ simplifies to $F_k = \Omega e_k^2$ with substitutions from equation (4), $\Omega_k = \gamma_k$ and $e_k = \sqrt{f_k}$ [12]. Equation (6) is solved using the Levenberg-Marquardt [13] method:

$$(\mathbf{H} + \lambda \mathbf{I})x^* = -b \tag{8}$$

$\mathbf{H} = \sum \mathbf{J}_k^T \Omega_k \mathbf{J}_k$ is the system matrix (Hessian), λ is a damping factor, x^* is the optimal timed-elastic band states and $b = \sum e_k^T \mathbf{J}_k$ is the error term. $\mathbf{J}_k$ represents the Jacobian of the error function, which is calculated by numerical approximation from linearization at the current solution [12].

When $\mathbf{H}$ is sparse, this equation is solved in a numerically efficient way via sparse Cholesky decomposition algorithms [14] and a-priori ordering. Thus, the sparsity of the TEB system matrix $\mathbf{H}$ makes it efficiently optimizable [11].

2 Approach

Extension to the TEB-based path-planner is required to recover the control inputs of the vehicle. This is achieved by the introduction of control input-dependent vertices into the hypergraph along with a way to systematically design dynamics violation costs without relying on hand-picked geometric parameters.

A rough trajectory estimate (in this case, a shortest Euclidean-distance path generated by a sampling-based planner) is used to initialize the optimization planner. The trajectory is parameterized as a tuple

$$B := (Q, U, \tau) \tag{9}$$

of discretized arrays of poses Q, time differences τ between poses (as in the original TEB specification), and corresponding control inputs U,

$$U = \{u_i\}_{i=0...n} \tag{10}$$

where u_i is the control input value corresponding to the i-th segment. Figure 1 demonstrates the hyper-graph structure.

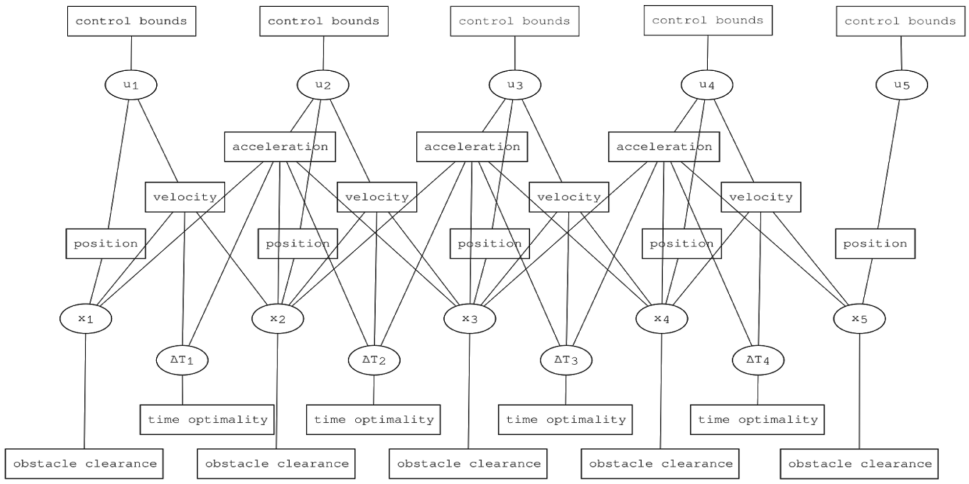


Figure 1: Hyper-graph structure. Poses (x), control, and time difference nodes are shown along with labels for hyper-edges.

Within the objective function $f(B)$, the classes of error included are:

- **Time Optimality**: A penalty proportional to each ΔT_i

$$f_{\text{time\_opt}}(\Delta T_i) = \Delta T_i \tag{11}$$

- **Obstacle Avoidance**: A penalty inversely proportional to the distance of the nearest obstacle to each q_i. A signed distance field is computed for the obstacle map and cost is assigned as follows with d_{min} as the minimum allowed distance to an obstacle and $d(q)$ the signed distance at configuration q.

$$f_{\text{obs}}(q_i) = \begin{cases} 0 & d(q_i) > d_{min} \\ \text{-(d(q}_i) - d_{min}) & \text{otherwise} \end{cases}$$

- **Control Bounds Violations**: A penalty proportional to the amount by which control bounds are violated for each u_i

$$f_{\text{u\_bounds}}(u_i) = \max(u_i - u_{\max}, 0) + \max(u_{\min} - u_i, 0)$$

- **Dynamics Constraints**: Penalties enforcing dynamics constraints for position, velocity, and acceleration of the vehicle (further detailed in section 2.1).

2.1 Penalties Associated with Dynamics Constraints

To remove the dependence on geometric information, a method of ensuring dynamics constraints based only on the current poses, vehicle controls, and independent vehicle parameters was developed. The error classes associated with dynamics constraints such as position, velocity, and acceleration constraints are specified as follows:

$$f_{position}(u_i, q_i) = q_i - h^{(0)}(u_i, q_i) \tag{12}$$

$$f_{velocity}(u_i, q_i, \dot{q}_i) = \dot{q}_i - h^{(1)}(u_i, q_i, \dot{q}_i) \tag{13}$$

$$f_{acceleration}(u_i, q_i, \dot{q}_i, \ddot{q}_i) = \ddot{q}_i - h^{(2)}(u_i, q_i, \dot{q}_i, \ddot{q}_i) \tag{14}$$

which compare the pose derivatives of the trajectory extracted by finite differencing values of pose and time nodes with those calculated using the corresponding control input node value, current pose, and further derivatives of the current pose. The pose derivatives on the left-hand side of the equation, calculated using

finite differencing as follows:

$$\dot{q}_i = \frac{q_{i+1} - q_i}{\Delta T_i} \tag{15}$$

$$\ddot{q}_i = \frac{2(\dot{q}_i - q_{i-1})}{\Delta T_{i+1} + \Delta T_i} \tag{16}$$

$$\vdots$$

$h^{(n)}(u_i, q_i, \dot{q}_i, \ddot{q}_i)$ propagates the nth pose derivative according to the current pose q_i, its derivatives, and the corresponding control input u_i. The specification for the $h^{(n)}(...)$ function is specific to each vehicle type and can be specified easily allowing for flexible use. This allows the enforcing of arbitrary vehicle dynamics. The Vehicle Models section below details the specifications of the $h^{(n)}(...)$ function for various vehicles simulated for evaluating the contribution.

2.2 Solving the Optimization Problem

The hyper-graph used in this contribution is optimized similarly to that in the original timed-elastic band problem. The requirement to optimize this graph efficiently enough for path-planning applications is that the graph remain sufficiently sparse. A sparse graph is one where the number of edges is much less than the possible number of edges between vertices. After the modifications to the hyper-graph this contribution presents, sparsity of the graph is sufficiently preserved if each edge in the graph depends only on few neighboring nodes. The addition of control input vertices and the control input-dependent edge penalties preserves the sparsity of the hyper-graph as each penalty edge still depends on few neighboring vertices.

The penalties preserve the sparsity of the hyper-graph as shown:

- Penalty edges related to each ΔT_i are dependent only on one vertex each.
- Penalty edges related to distance to nearest obstacle depend only on one position vertex q_i each
- Penalties enforcing dynamics constraints depend on a cluster of pose vertices, control vertices, and time vertices that has size $O(n)$ where n is the highest derivative specified in the vehicle model.

- Penalty edges for control bound violations depend only on one control vertex u_i each

Since sparsity is retained, Levenberg-Marquardt with sparse Cholesky decomposition algorithms can optimize the graph efficiently [14] . Indeed, results show optimization time in the order of seconds.

3 Results

3.1 Implementation

The hyper-graph structure was implemented in C++ using a custom datastructure with functionalities for adding and deleting vertices, updating weights of edges and vertices, and updating the hypergraph structure accordingly. Interchangeable edge and vertex types were implemented.

The Open Motion-Planning Library (OMPL) [15] provides implementations of many sampling-based motion planning algorithms. It was used to provide an initialization trajectory for the optimization. The initialization trajectory was calculated using a Probabilistic Roadmap [3] planner with no information about vehicle dynamics or vehicle orientation information. Obstacle, start, and goal pose nodes remain fixed during optimization. All pose orientations were initialized to the goal orientation. All control inputs were initialized to the same value, an average of the minimum and maximum value for the control input.

Arriving at a feasible trajectory requires that during the optimization, the trajectory maintains some level of resolution such that an update does not result in a trajectory with two consecutive poses that are too close or too far apart. As such, it was necessary to implement graph resizing functionality. The hyper-graph resizes by adding or removing nodes as needed when any two consecutive poses exceed a set minimum or maximum distance from each other. The hyper-graph adds a pose node that is obtained by interpolating between the two poses and automatically makes the necessary connections. Since these connections affect few neighboring nodes, the rest of the graph is unchanged, and therefore graph resizing is not exceedingly expensive as long as only few nodes are added or deleted.

For optimization of the hypergraph, the open-source framework for sparse system solvers ”g2o” [16] was used. The results of the tests were visualized using a custom-built GUI.

3.2 Vehicle Models

The contribution was tested with simulated vehicle models with varying dynamics. In line with the main goal of this project, the planner is able to accept simple parametric vehicle descriptions specified in terms of their dynamical equations of motion. The specification of the models is done by outlining the pose and control input types, and the definition of the $h^{(n)}$ function for the vehicle type, along with independent vehicle parameters. The dynamics models used in testing the contribution are outlined below.

3.2.1 Double Integrator System

The 3D double-integrator system is a simple system where the control inputs are the target accelerations. For a 3D double integrator, the poses q_i are defined by the Cartesian coordinates of the system and the control inputs are the target accelerations.

$$q_i = [x, y, z] \tag{17}$$

$$u_i = [a_x, a_y, a_z] \tag{18}$$

The dynamics specified by the $h^{(n)}$ function are:

$$h^{(2)}(u_i, \cdot, \cdot, \cdot) = u_i$$

3.2.2 Dubins Car-like System

The Dubins car model [17] describes a car that can either turn right at maximum, turn left at maximum, or drive forward. The poses for a Dubins car are defined by the Cartesian coordinates x, y, z and the yaw angle θ.

$$q_i = [x, y, z, \theta] \tag{19}$$

The control input is determined by two parameters: the steering angle ϕ and acceleration a of the vehicle.

$$u_i = [\phi_i, a_i] \tag{20}$$

The model is parameterized by two independent values, the wheelbase L of the vehicle, and d, the distance from the ground.

$$p = [L, d] \tag{21}$$

The dynamics specified by the $h^{(n)}$ function are:

$$h^{(0)}(\cdot, q_i) = [q_i^{(x)}, q_i^{(y)}, d, q_i^{(\theta)}] \tag{22}$$

$$h^{(1)}(u_i, q_i, \dot{q}_i) = [|\dot{q}_i| \cos(q_i^{(\theta)}) |\dot{q}_i| \sin(q_i^{(\theta)}), \phi_i, \text{atan2}(L, \frac{|q_i^{(x,y)}|}{q_i^{(\theta)}}] \tag{23}$$

$$h^{(2)}(u_i, q_i, \cdot, \ddot{q}_i) = [\ddot{q}_i{}^{(x)} + a_i \cos(q_i^{(\theta)}), \ddot{q}_i{}^{(y)} + a_i \sin(q_i^{(\theta)}), 0, \ddot{q}_i{}^{(\theta)}] \tag{24}$$

3.2.3 Point Mass Airplane

The point mass airplane model requires a five-dimensional pose which includes a 3D Cartesian coordinate position and orientation specified by yaw ψ and pitch γ [18].

$$q_i = [x, y, z, \psi, \gamma] \tag{25}$$

The control input is composed of the roll angle ϕ, forward thrust force magnitude F_T and lift coefficient C_L.

$$u_i = [F_T, C_L, \phi] \tag{26}$$

Parameters for the airplane model are gravity g, air density ρ, aircraft mass m, wing surface area S, constant $K = \frac{1}{\pi e AR}$ where e is the wing efficiency and AR the aspect ratio, and zero-lift drag coefficient C_{D_0} [18].

The dynamics of the point mass airplane are given by the following equations:

$$h^{(1)}(u_i, q_i, \dot{q}_i) = \begin{bmatrix} |\dot{q}_i| \cos(q_i^{(\gamma)}) \cos(q_i^{(\psi)}) \\ |\dot{q}_i| \cos(q_i^{(\gamma)}) \sin(q_i^{(\psi)}) \\ |\dot{q}_i| \sin(q_i^{(\psi)}) \\ -\frac{F_L(u_i^{(C_L)}, |\dot{q}_i^{(x,y,z)}|, p^{(S)}, p^{(\rho)}) \sin(u_i^{(\phi)})}{p^{(m)} |\dot{q}_i^{(x,y,z)}| \cos(\dot{q}_i^{(\gamma)})} \\ \frac{F_L(u_i^{(C_L)}, |\dot{q}_i^{(x,y,z)}|, p^{(S)}, p^{(\rho)}) \cos(u_i^{(\phi)})}{p^{(m)} |\dot{q}_i^{(x,y,z)}|} - \frac{p^{(g)}}{|\dot{q}_i^{(x,y,z)}|} \cos(\dot{q}_i^{(\gamma)}) \end{bmatrix} \tag{27}$$

$$h^{(1)}(u_i, q_i, \cdot, \ddot{q}_i) = \begin{bmatrix} \ddot{q}_i + \cos(q_i^{(\gamma)}) \cos(q_i^{(\psi)}) a_{\text{target}} \\ \ddot{q}_i + \cos(q_i^{(\gamma)}) \sin(q_i^{(\psi)}) a_{\text{target}} \\ \ddot{q}_i + \sin(q_i^{(\psi)}) a_{\text{target}} \\ \ddot{q}_i^{(\psi)} \\ \ddot{q}_i^{(\gamma)} \end{bmatrix} \tag{28}$$

$$a_{\text{target}} = \frac{u^{(F_T)}}{p^{(m)}} + \frac{F_D(u^{(C_L)}, |\dot{q}_i^{(x,y,z)}|, p^{(S)}, p^{(\rho)}, p^{(C_{D_0})}, p^{(K)})}{p^{(m)}} - p^{(g)} \sin(q^{(\gamma)}) \tag{29}$$

$$F_L(C_L, v, S, \rho) = \frac{1}{2} \rho v^2 S C_L \tag{30}$$

$$F_D(C_L, v, S, \rho, C_{D_0}, K) = \frac{1}{2} \rho v^2 S (C_{D_0} + K C_L^2) \tag{31}$$

3.3 Testing

The environments simulated for testing are shown in Figure 2. The path planner was tested for each of the simulated vehicle dynamics in each testing environment. Results are shown for the following experiments: traversing the 2D maze environment with the Dubins car, the 3D walls environment with the double integrator, the 3D walls environment with the point-mass airplane, and the crowded environment with the airplane.

Figure 3 shows the optimized and initial trajectories of each of these test cases. In this figure, it can be seen that the initial trajectory does not take into account vehicle dynamics or pose orientation, evidenced by the sharp and geometric look of direction changes in the initial paths, while the optimized trajectory does both. The optimized trajectories can be followed by simulations of the vehicle models exactly.

Figure 4 shows the value of the total error given by the objective function $f(B)$ over the optimization time for each of the test cases. The total error is reduced to below an order of magnitude within 60 seconds for the airplane, and much faster, within the order of a few seconds, for the double integrator and Dubins car. This is expected as the airplane has a much more non-linear dynamics model. The optimization time calculated includes visualization and I/O in the optimization loop and is therefore simply for evaluation of convergence, rather than a rigorous measurement of the runtime of the algorithm.

Figure 5 shows the plots for the test case of the double integrator system in the 3D walls environment. The errors for each penalty category except time optimality are below 10^{-6} with time optimality remaining low, below 1. This is expected, since a certain amount of time optimality error will always be present due to the value of the error being proportional to the time required to traverse the trajectory, whereas the other penalties are hard constraints associated with a high cost multiplier in the objective. Spikes in these errors are apparent which correspond to resizing of the graph, which momentarily increases errors as nodes and edges are added and deleted. The velocity of the system follows the shape of the trajectory, decreasing around turns and increasing along straighter segments. Acceleration in each direction follows what is expected for the shape of the trajectory.

Figure 6 displays significant plots for the Dubins car in the 2D maze environment experiment. On the left of the figure are plots for each category of error making up the total objective. All types of error remain below the order of 10^{-6}, with the exception of the error associated with time optimality, which is on the order of 10. The figure also shows the plot for the velocity of the car along the optimized trajectory and the resulting values for its two control inputs: linear acceleration and steering angle. The velocity of the car along the path aligns with the shape of the trajectory. The car speeds up to its maximum speed of $5m/s$, remains there for the majority of the trajectory time, and slows down as it nears the goal. The control inputs, linear acceleration and steering angle, also qualitatively correspond with the output trajectory. The car starts the trajectory at its maximum acceleration of $2m/s^2$, then levels off close to zero around the time of the

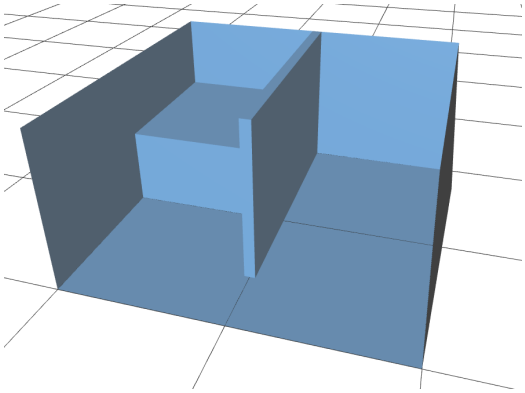
velocity peak, then decelerates. The steering angle plot shows maximum steering around corners in the environment. The optimization seems to recover control inputs that resemble a bang-bang policy, which is the expected optimal control policy of the vehicle.

Figure 7 shows these plots for the airplane traversing across the crowded environment. As in previous experiments, the errors for each category of penalty, except time optimality, are very close to zero. The time optimality error is also small, on the order 10^{-2}. The velocity plot and resulting control inputs qualitatively correspond with expectation in this case as well. Velocity is constant and high for the majority of the trajectory time, decreasing sharply to stop at the goal position. The output thrust force F_T increases to a maximum at the beginning of the trajectory, then gradually decreases to zero. The lift coefficient C_L increases and decreases along more prominent turns in the trajectory. The roll angle ϕ follows the shape of the horizontal component of the trajectory. As the trajectory is close to a straight line with slight vertical and horizontal turns, this corresponds with expectations.

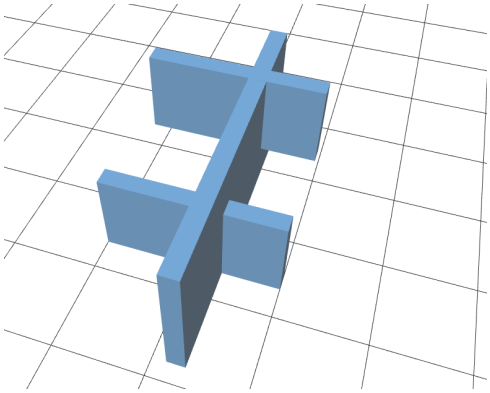
Figure 8 shows the same plots for the airplane traversing the 3D walls environment. As in the previous experiments, the errors associated with each penalty category remain below several orders of magnitude, with the exception of time optimality, which is within an order of 10^{-2}. Small spikes occur within each of these error plots which can be attributed to moments of resizing of the graph. The velocity plot and resulting control inputs qualitatively correspond with expectations. The velocity increases sharply along straight segments and decreases at turning points in the environment. The resulting forward thrust force magnitude F_T increases as the trajectory of the plane turns horizontally and lifts over a wall. The lift coefficient C_L and the roll angle ϕ increase to a maximum at the beginning and end of the trajectory, decrease as the plane dips along a straight path, and increase to a maximum around the horizontal turn in the trajectory.

4 Conclusion and Future Work

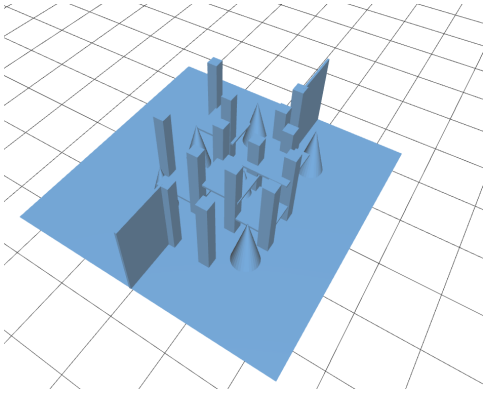
The goal of this project was to design a motion planner well-suited for application to these less traditional planning problems by avoiding having to sacrifice flexibility and expressiveness in the specification of the target vehicle's dynamics model. The results from simulation conclude that the approach presented can reliably produce feasible trajectories and accurately return control inputs. The planner is able to accept



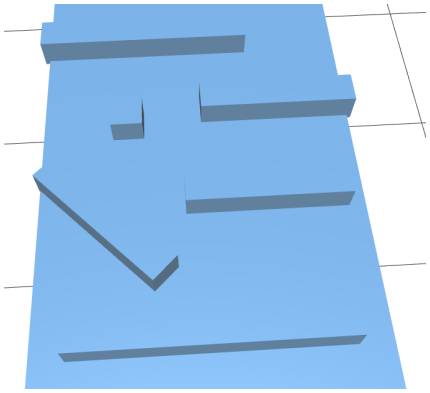
(a) Simple environment with a corner and elevated surface.



(b) Simple environment with high walls to be flown over



(c) Crowded environment with many obstacles



(d) Maze environment with many corners

Figure 2: Environments simulated for testing

simple and elegant parametric vehicle descriptions in terms of their dynamical equations of motion while showing good performance in the evaluation carried out so far.

The innovation of this planner is to augment the timed elastic band with controls and parameter information. The TEB approach, which is originally unable to recover controls and whose vehicle models rely on hand-picked geometric features of admissible paths, is extended in two major ways: control input vertices are introduced, and a systematic way of designing the dynamics violation costs is introduced. It has been demonstrated that the planner exhibits a sparse system structure as the TEB, and can therefore also be solved efficiently using sparse nonlinear optimization solvers. Experiments have shown that the planner can produce high-quality trajectories in reasonable runtime. In a similar manner to the TEB, the planner is also suitable for higher dimensional spaces.

Future work could focus on a more rigorous evaluation of this method. Optimality guarantees as well as a thorough investigation into the method's efficiency should be explored. Efficiency can be improved through various means such as providing analytical Jacobians to the optimization algorithm. A clear extension of this method is the joint optimization of additional parameters, such as the handpicked vehicle parameters included in the vehicle models used for testing. These parameters could be incorporated as nodes in the hyper-graph with corresponding hyper-edges and optimized in synchronization with the trajectory and control inputs. Methods to maintain sparsity of the graph with the addition of parameter nodes will need to be explored.

References

[1] A. Abrams and R. Ghrist. Finding topology in a factory: Configuration spaces. *The American Mathematics Monthly*, 109:140–150, February 2002.

[2] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[3] D. Aarno, D. Kragic, and H. I. Christensen. Artificial potential biased probabilistic roadmap method. In *Proceedings IEEE International Conference on Robotics & Automation*, 2004.

[4] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University, October 1998.

[5] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, June 2011.

[6] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *CoRR*, abs/1105.1186, 2011.

[7] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*. IEEE, May 2009.

[8] David G Luenberger. *Introduction to dynamic systems : theory, models, and applications*. Wiley, 1979.

[9] R C Coulter. Implementation of the pure pursuit path tracking algorithm. 1992.

[10] J. Kim and J. P. Ostrowski. Motion planning of aerial robot using rapidly-exploring random trees with dynamic constraints. In *Proceedings IEEE International Conference on Robotics & Automation*, 2003.

[11] Christoph Rosmann, Wendelin Feiten, Thomas Wosch, Frank Hoffmann, and Torsten Bertram. Efficient trajectory optimization using a sparse model. In *2013 European Conference on Mobile Robots*. IEEE, September 2013.

[12] Christoph Rösmann, Wendelin Feiten, Thomas Wösch, Frank Hoffmann, and Torsten Bertram. Trajectory modification considering dynamic constraints of autonomous robots. *Proc. 7th German Conference on Robotics, Munich, Germany*, page 74–79, 2012.

[13] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2(2):164–168, July 1944.

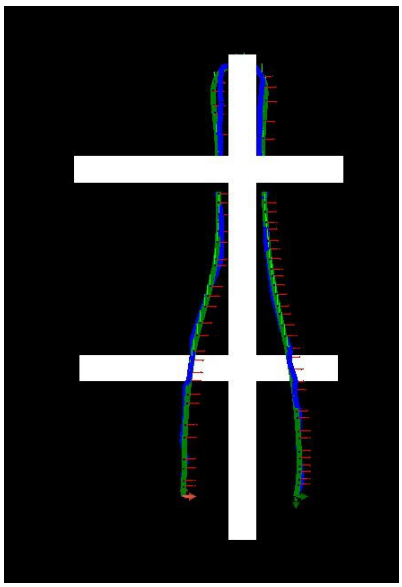
[14] Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887. *ACM Transactions on Mathematical Software*, 35(3):1–14, October 2008.

[15] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. https://ompl.kavrakilab.org.

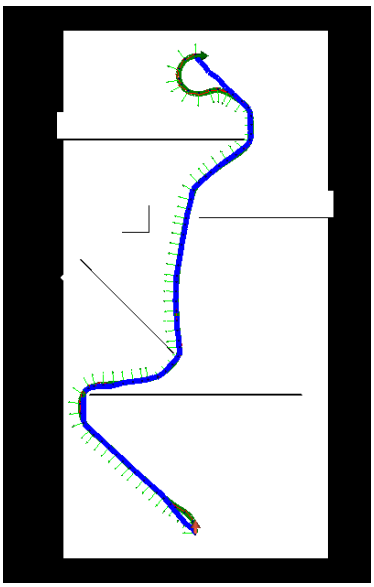
[16] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011.

[17] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79:497–516, 1957.

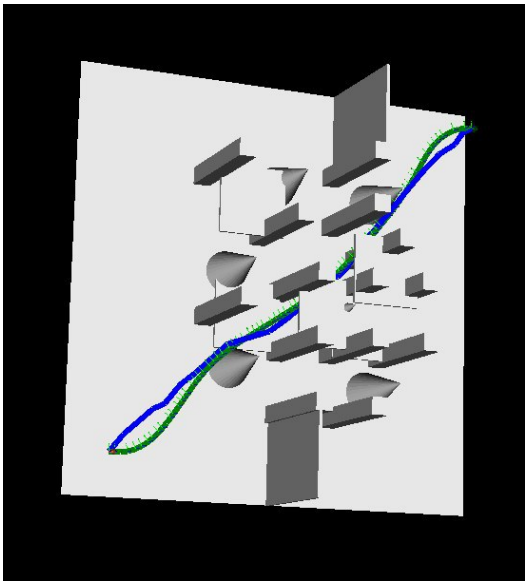
[18] Lesley A. Weitz. Derivation of a point-mass aircraft model used for fast-time simulation. Technical report, The MITRE Corporation, 2015.



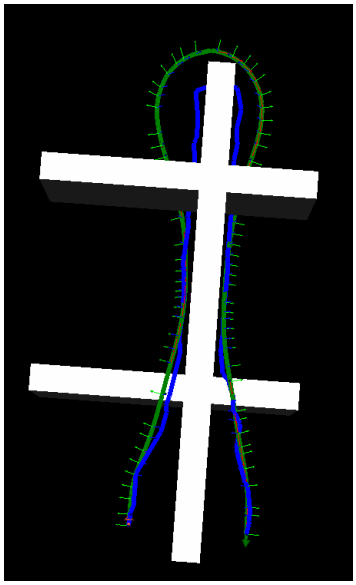
(a) Double integrator system in 3D walls environment



(b) Dubins car in 2D maze environment.

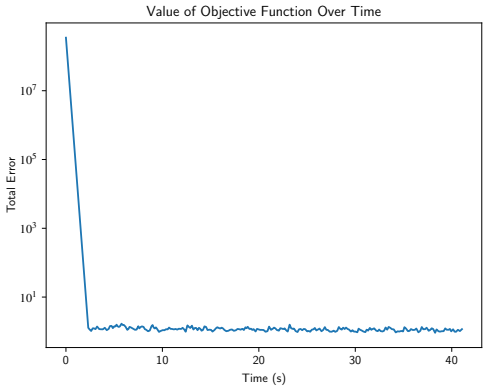


(c) Plane in crowded environment

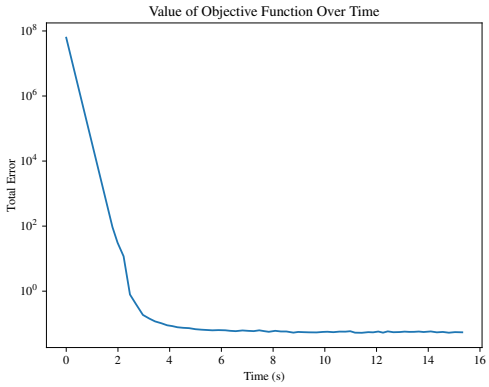


(d) Plane in 3D walls environment

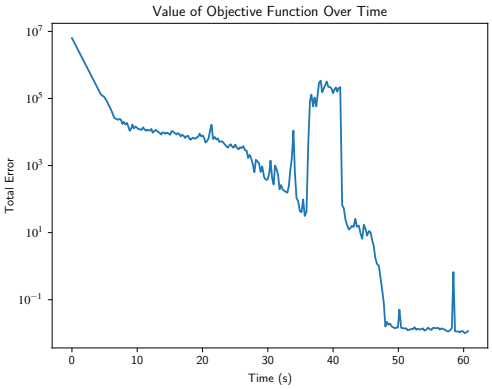
Figure 3: Trajectories of the experiments. Green shows the optimized trajectory, blue shows the initialization trajectory. Arrows indicate orientation at a point.



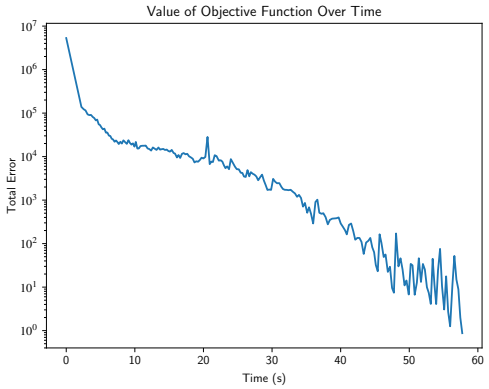
(a) Double integrator system in 3D walls environment



(b) Dubins car in 2D maze environment.



(c) Plane in crowded environment



(d) Plane in 3D walls environment

Figure 4: Value of the total objective function over optimization time for each test case.

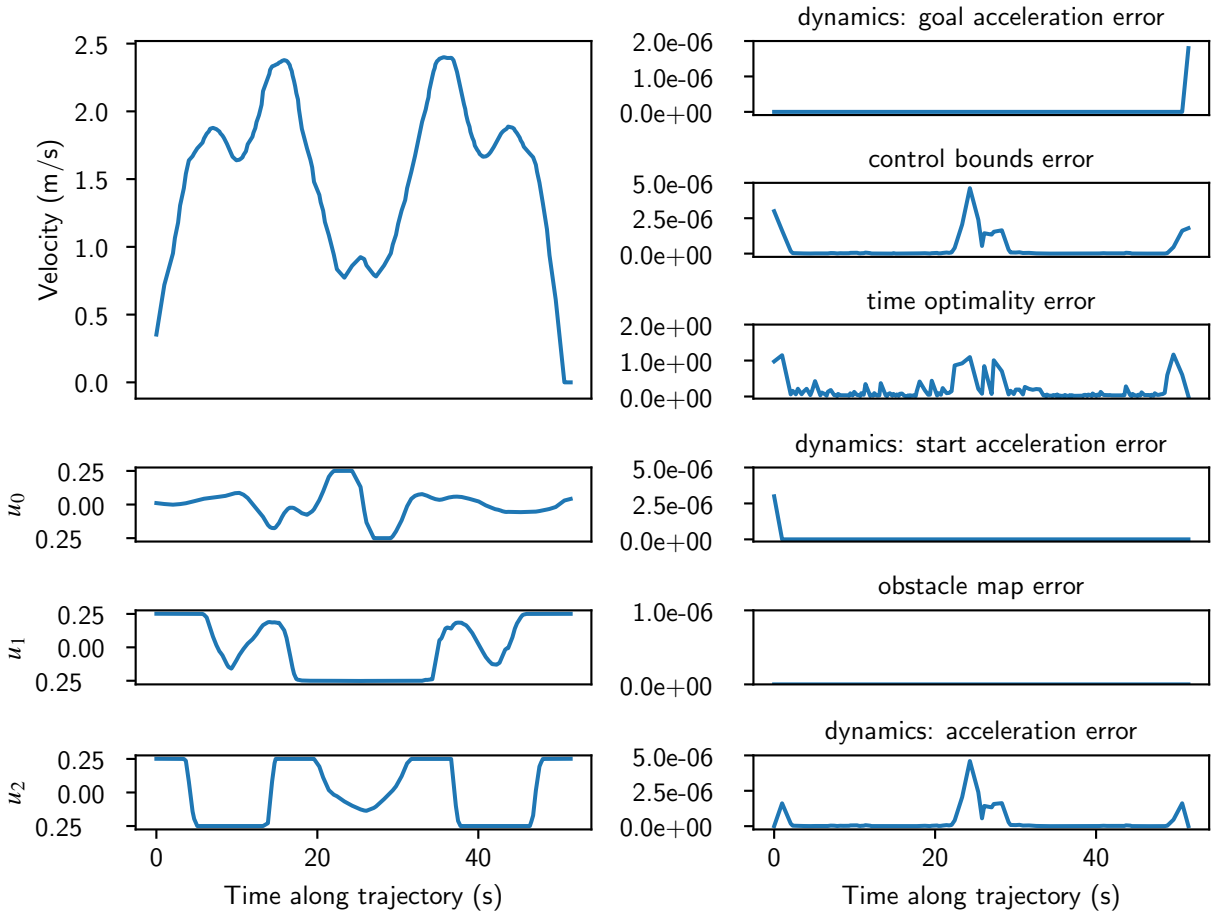


Figure 5: Errors by category for the double integrator system in the 3D walls environment (right). The control inputs u_0, u_1, u_2 correspond to the acceleration in the x, y, and z directions respectively. Velocity of the vehicle along the trajectory is shown in the top left with resulting control input values below.

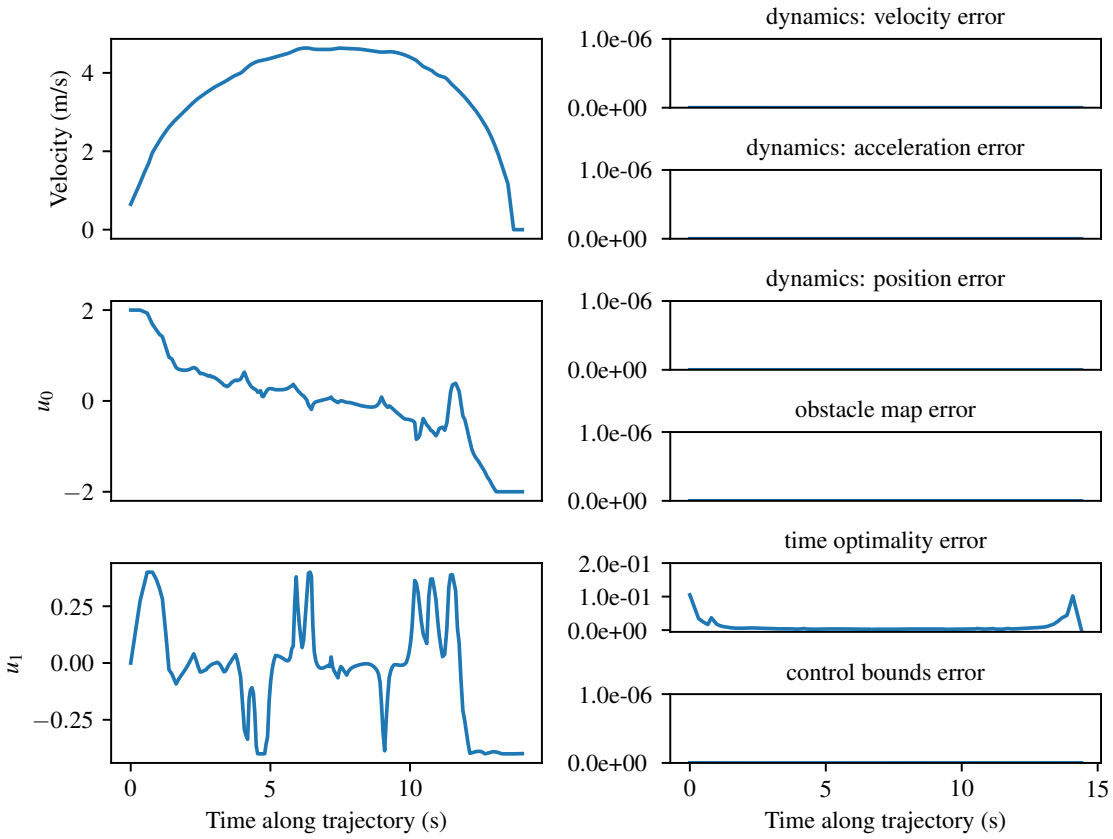


Figure 6: Velocity and control inputs (right) for the optimized trajectory for the Dubins car in the 2D maze environment. The control input u_0 corresponds to linear acceleration. The control input u_1 corresponds to steering angle. The error plots for each category in the sum of the objective function are shown (left). Each category of error remains below $1e-6$, with the exception of time optimality.

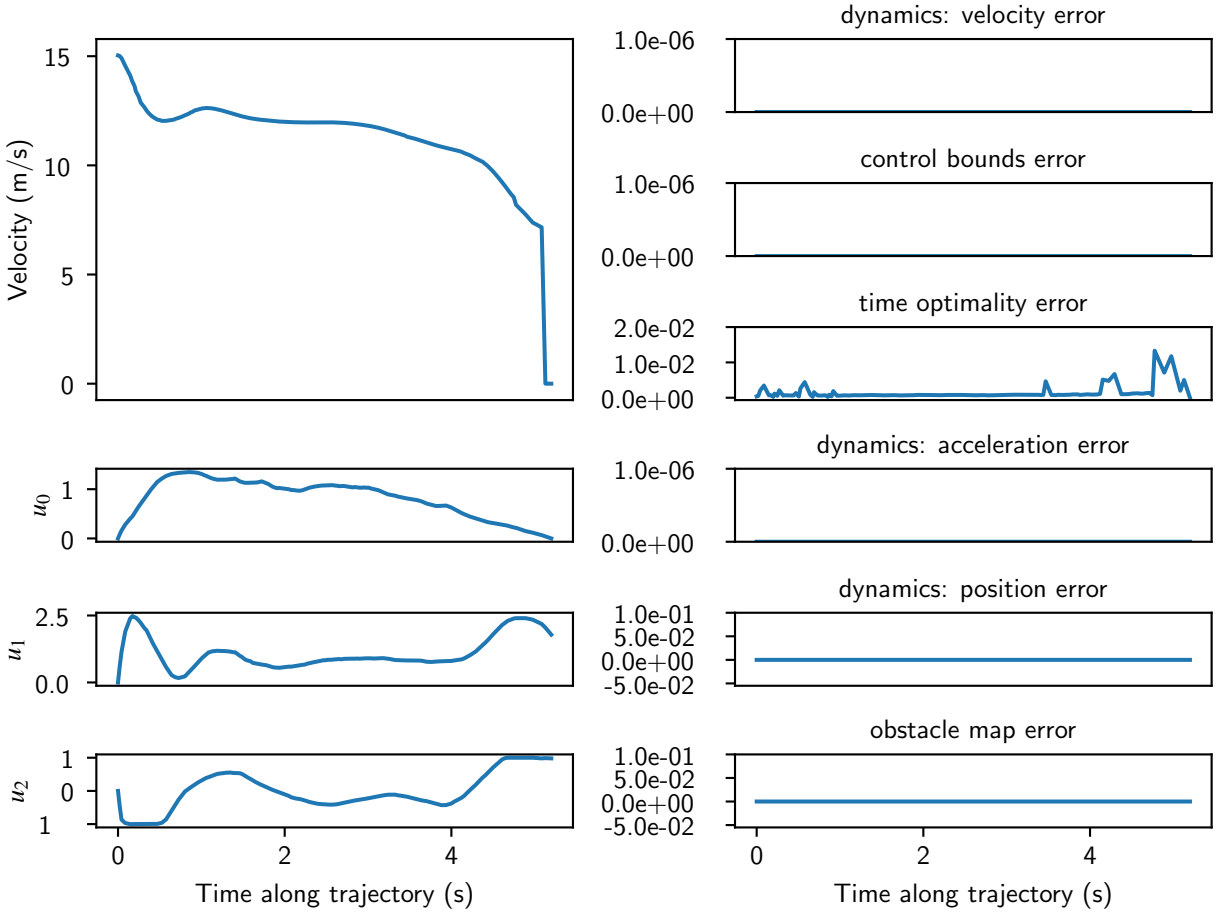


Figure 7: Errors by category for the plane in the crowded environment (right). The control input u_0 corresponds to the forward thrust force magnitude F_T, u_1 to the lift coefficient C_L, and u_2 to the roll angle ϕ. Velocity of the vehicle along the trajectory is shown in the top left with resulting control input values below.

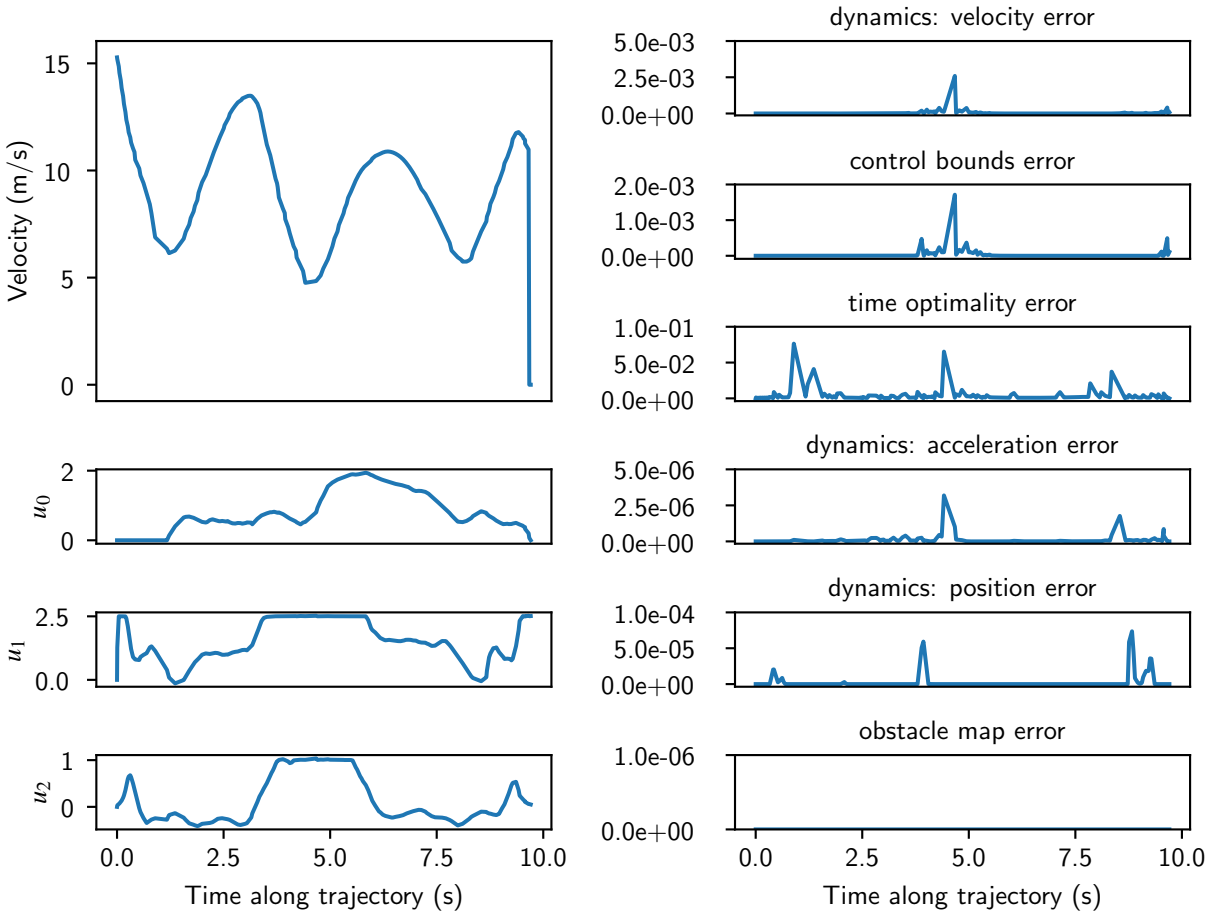


Figure 8: Errors by category for the plane in the 3D walls environment (right). The control input u_0 corresponds to the forward thrust force magnitude F_T, u_1 to the lift coefficient C_L, and u_2 to the roll angle ϕ. Velocity of the vehicle along the trajectory is shown in the top left with resulting control input values below.