The Rapidly exploring Random Tree Funnel Algorithm

1st Ole Petter Orhagen
Department of Physics
University of Oslo
Oslo, Norway
olepor@matnat.uio.no

2nd Marius Thoresen
Defence Systems Division
Norwegian Defence Research Establishment, FFI
Kjeller, Norway
marius.thoresen@ffi.no

2nd Kim Mathiassen
Norwegian Defence Research Establishment, FFI
Dept. Technology Systems, University of Oslo
Kjeller, Norway
kim.mathiassen@ffi.no

***Abstract*—This paper shows the feasibility of combining robust motion primitives generated through the Sums Of Squares programming theory with a discrete Rapidly exploring Random Tree algorithm. The generated robust motion primitives, referred to as funnels, are then employed as local motion primitives, each with its locally valid Linear Quadratic Regulator (LQR) controller, which is verified through a Lyapunov function found through a Sum Of Squares (SOS) search in the function space. These funnels are then combined together at execution time by the Rapidly-exploring-Random-Tree (RRT) planner, and is shown to provide provably robust traversal of a simulated forest environment. The experiments benchmark the RRT-Funnel algorithm against an RRT algorithm which employs a maximum distance to the nearest obstacle heuristic in order to avoid collisions, as opposed to explicitly handling uncertainty. The results show that employing funnels as robust motion primitives outperform the heuristic planner in the experiments run on both algorithms, where the RRT-Funnel algorithm does not collide a single time, and creates shorter solution paths than the benchmark planner overall, although it takes a significantly longer time to find a solution.**

***Index Terms*—Collision avoidance, Motion planning, Nonlinear control systems, Robot control**

I. Introduction

Motion planning concerns the problem of finding a dynamically feasible path from an initial configuration to a defined end state in a safe manner. In order for a motion planner to handle real world motion planning tasks it needs to handle the uncertainty that comes with a real-life planning problem. This is especially difficult for nonlinear dynamical systems. Knowledge of the system's state, the environment and the dynamics of the system are all uncertain to some degree. Limited measurement precision and use of imperfect world and vehicle models will therefore leave error terms in the state and world estimations. Thus, in order for a planner to guarantee safe traversal through a real world environment, a motion planner needs to handle uncertainties.

In the face of uncertainty, some planners choose to ignore these error sources and instead apply heuristics such as maximizing the distance to the obstacles in the environment or setting a fixed safety radius [1]. However, this adds the disadvantage that the plans can become overly conservative. Explicitly handling the uncertainties in the planning stage enables the planner to employ more aggressive maneuvers, such as going through two obstacles that are close together, as opposed to going around the difficult area. If uncertainties are accounted for, going straight through is an acceptable maneuver for a planner that has guarantees on the whereabouts of the dynamical system, and hence is not afraid to get close to an obstacle. This means that a robust motion algorithm can perform more aggressive maneuvers than one that is inherently conservative about its environment and maneuvers [2].

In motion planning, there is a separation between global and local methods [1], where global methods assumes a known map of the environment and local methods assumes only local knowledge. Thus, global methods can plan paths from the start state to the goal, whereas local methods can only plan a part of the way. As local methods explicitly consider the end goal during planning, they are more susceptible to getting trapped in local minima.

Previous efforts to handle uncertainties in global path planning include using the Rapidly-exploring Random Tree (RRT) algorithm with robust motion primitives. This approach is taken by Tedrake [3], where a tree consisting of robust Linear-Quadratic Regulator (LQR) controllers are created, in order to guarantee safe traversal of the planning environment. Van den Berg [4] has a similar approach of creating motion primitives where a LQR is used, and a Gaussian process is used to model uncertainty, thus allowing robust control along with an uncertain system. Luders generate robust motion primitives in real-time for an RRT* algorithm through chance constraints in [4], but also ensures asymptotic optimality. Another robust extension to the RRT algorithm can be found in Melchior [5], where each extension to the tree is treated as a stochastic process, and simulated multiple times, and since pruned based on the expected probability of successful execution.

A recent approach to the local path planning problem has been to use Sum of Squares (SOS) theory with motion planning. As mentioned above, this is the approach taken by Tedrake in [3]. In [6], Majumdar expands upon this idea and seeks to limit the size of the area in which a controller will take a dynamic system, while at the same time giving guarantees of safe traversal. This enables real-time motion planning in highly complex environments with advanced vehicle models, as long as the controllers are generated off-line. The offline generation is a requirement due to the time they take to generate. The

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1109/ICMRE54455.2022.9734089

current research on the topic is limited, but interesting due to the formal general robustness guarantees it provides.

However, the method is a local method, and thus has a limited planning horizon. In [6], a simulated experiment with large-scale environments is conducted, but only by repeating the local process several times. If we instead can use the theory and framework introduced here and combine it with a global planner, we can achieve longer planning horizons and thereby reduce likelihood of getting trapped in local minima.

This paper builds upon the work done on verifying regions of attraction — referred to as funnels — for nonlinear dynamical systems through the use of Lyapunov functions. Verification is achieved for a polynomial system through the use of SOS programming. We combine the theory of funnels with a discrete RRT motion planner in order to provide a provably robust global motion planner for a nonlinear dynamical system. To the best of the authors' knowledge, this is the only known RRT algorithm which implements verified reachable sets generated through SOS programming as discrete motion primitives as extension operators for the an RRT algorithm. The novel method is then compared with a RRT algorithm that maximize the distance to the nearest obstacles, in order to show which handles the planning task best. The result of this work is the RRT-Funnel motion planning algorithm, a discrete motion planning algorithm which can guarantee safe passage through an obstacle space, given the assumption that the uncertainty in the system is bounded. The funnel definitions in this paper is taken from a series of articles on funnels [3], [6]–[9], with the main focus being on [6].

II. Method

This paper develops the RRT-Funnel algorithm, by two means: First it employs robust motion primitives generated by the the SOS programming framework based on the work in [6], and second, deploy these funnels as robust motion primitives in a discrete RRT robust motion planner based on [10]. Using robust motion primitives has several advantages. Firstly, they handle uncertainty, and thus, as long as the uncertainties in the system are akin to the assumptions on the incoming uncertainty parameters, the dynamical system will not leave the funnel, and hence if the funnel is not in collision, neither will the system be. Secondly, as the motion primitives are robust, there is no need for more conservative maneuvers and heuristics, such as maximizing the distance to an obstacle, which is a naive way for motion planners to handle uncertainty. Since the primitives are robust, the system might as well choose a primitive that is close to an obstacle, as one that is far away, since the funnel is guaranteed to be collision-free in both cases. This means that a robust motion algorithm can perform more aggressive maneuvers than one that is inherently conservative about its environment and maneuvers [2].

A. Problem Statement

Given the nonlinear dynamical system

$$\dot{\mathsf{x}} = f\big(\mathsf{x}(t), \mathsf{u}(t), w(t)\big), \tag{1}$$

with state $\mathsf{x}(t) \in \mathbb{R}^3$, $\mathsf{u}(t) \in \mathbb{R}$, and $w(t) \in \mathbb{R}$. Then create a set of robustly verified discrete motion primitives through the use of SOS programming to verify the reachable set for each of the trajectories in a base set of trajectories T_0. Next, apply these trajectories, with the robust reachable set surrounding them as motion primitives for a discrete RRT motion planning algorithm and compare the results to a RRT algorithm which does not employ the reachable sets around the base trajectories, but instead relies on maximizing the distance to the nearest obstacles, in order to handle uncertainty.

B. Funnels

The uncertainty guarantees in this paper is given through the creation of *funnels*. Funnels are the parameterizations of the *finite time reachable sets* for the dynamic system in 1.

This means that a Funnel holds all the states the dynamical system can be in during a planning task. Mathematically the reachable set of the system is defined as

$$\mathsf{x}(0) \in \mathcal{X}_0 \implies \mathsf{x}(t) \in F(t), \forall t \in [0, T],$$

where $\mathcal{X}_0$ is the set of initial conditions, $[0, T]$ the time interval, and $F(t)$ is the set of states that the system can be in at time t [6]. Although this paper concerns itself with approximating the reachable set through *Lyapunov* functions, a useful analogy is imagining the funnel created through a *Monte-Carlo* simulation, where the funnel is the set of all the paths traversed by the dynamic system.

1) Generating Trajectories: In order to verify the robust regions surrounding a trajectory, first the trajectories themselves have to be created. Generating optimal trajectories is a rich field in the motion planning literature [11]. The initial trajectories can be generated by many different methods, however the *direct collocation method* [12] suited the needs of this paper best. It was chosen as it builds locally optimal trajectories from a discrete set of sampled points along a sought trajectory, which is beneficial for the discrete funnel verification employed. For this problem the cost function chosen for the solver to minimize is:

$$J = \int_0^T \left[1 + \mathsf{u}_0{}^T R \mathsf{u}_0\right] \mathrm{d}t, \tag{2}$$

where $R = 1$, because it will minimize the system input, and thus give a smooth output trajectory [8].

2) Initializing the Funnel Calculations: The funnel calculation algorithm has to be initialized with a candidate Lyapunov function. In the same way as in Majumdar [8], the funnel generation algorithm will be initialized with a Time Varying Linear Quadratic Regulator (TV-LQR) as the initial Lyapunov function employing a cost function of the form

$$\begin{aligned} J = {} & \mathsf{x}^T(t_f) F(t_f) \mathsf{x}(t_f) \\ & + \int_{t_0}^{t_f} \left(\mathsf{x}^T Q \mathsf{x} + \mathsf{u}^T R \mathsf{u} + 2\mathsf{x}^T N \mathsf{u}\right) \mathrm{d}t, \end{aligned} \tag{3}$$

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1109/ICMRE54455.2022.9734089

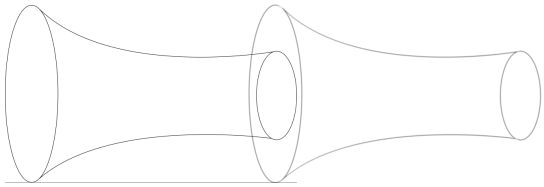

Fig. 1. Two funnels that can be successfully composed, as the outlet of the first one is fully contained in the inlet of the second. The system start position is the left side of the figure and the goal position to the right.

3) Generating Funnels around the Initial Trajectories: With the set of initial trajectories generated from the *direct collocation method* in II-B1, using SoS optimization it is possible to generate an area around the trajectories which are able to handle a bounded uncertainty parameters. This is done through the SOS optimization framework from Majumdar [6]. This method takes as input the initial trajectories, and an initial Lyapunov function, and outputs a verified region around the trajectories which are guaranteed to be collision free as long as the system uncertainties are bounded.

4) Sequential Funnel Composition: Once funnels have been generated as discrete motion primitives, it is time to handle the overarching goal of creating a plan consisting of multiple funnels chained from start to finish, so that safe traversal can be guaranteed along the given planning trajectory. In order for two funnels to create one extended motion primitive from multiple smaller primitives, the funnels in use must be composable. In order for two funnels to be composable, the outlet of one funnel needs to be completely contained within the inlet of the other [8]. An abstract pictorial representation of two funnels composed together can be seen in Fig. 1 to emphasize this observation.

Since the funnels can be shifted freely around the configuration space along the cyclic coordinates of the system to create new motion primitives [8], the funnels can be composed together by the RRT algorithm in the planning environment and create a guaranteed robust motion plan through chaining multiple verified funnels together into a longer and verified motion primitive.

C. RRT

With the basic framework for dealing with funnels as motion primitives constructed, it is time to build the RRT part of the RRT-Funnel algorithm. The reason for basing the global path planning framework on the RRT motion planning algorithm is twofold. Firstly, it has the ability to quickly expand deep into the search-space, and then later progress towards a finer sampling, which is valuable as it avoids local minima. Secondly, the RRT algorithm is extensible to larger state spaces, and the RRT-Funnel algorithm can therefore be adapted to fit a wide range of dynamical systems. Extending the RRT algorithm to a larger state space requires modifying the three main components; the probability distribution to sample from, the distance metric for the nearest neighbor and the extension step. The RRT algorithm is beneficial as most of the complexity accompanied with the planning problem (like uncertainty and controller calculations) is handled by the SOS framework, and hence the RRT algorithm need only concern itself with stacking one robust motion primitive after the other without any concern for the complexities associated with uncertain dynamical systems.

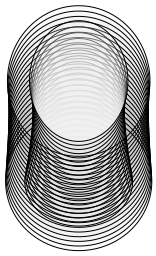

Fig. 2. The original funnel created from the point model, with a funnel expanded by a radius of 0.1 surrounding it.

1) Distance in Configuration Space: The RRT-Funnel algorithm will use the same metric for both the closest node and the extend operation on the funnel graph. The metric chosen is a modified Euclidean metric which weights the angle θ depending on how close the dynamical system is to the final configuration, and is defined as

$$\rho(\mathsf{x}_1, \mathsf{x}_2) = w_1 \|X_1 - X_2\| + w_2 f(\theta_1, \theta_2),$$

where $\|X_1 - X_2\|$ is the standard Euclidean metric, f is a positive scalar function giving the angle between headings [13]. The rotations and distance is then scaled relative to the translation distance by w_1 and w_2. Which helps solve some of the problems with the Euclidean distance metric [10].

2) Optimize the Selection Step: It is helpful to associate some structure along with the funnels which the RRT-Funnel algorithm employs at the planning stage [8]. This is because not all funnels are compatible, as one funnels outlet might not fit into the outlet of another. Thus the funnels generated can be imagined as a directed graph, where an edge from funnel A to funnel B means that (A, B) are composable. Thus a brute-force search for compatible funnels at planning time is eliminated. Hence the funnels are organized into a graph structure $\mathcal{G}$ where each funnel is an edge in the graph. Therefore no test to see if two funnels in the motion primitive set are compatible is needed.

3) Expanding the Size of the Funnels: In general the funnels generated are computed only for the point model in Eq. (1), and hence, in order to run the algorithm with a model of some defined size, the funnels have to be expanded by the largest radius of the model. Since the funnels are ellipsis around the point at the trajectory that they verify, the funnels can be expanded by any radius with a linear transformation. An expansion of a funnel around the point model used in this paper can be seen in Fig. 2.

III. Experiments

The experiments will run the RRT-Funnel against a benchmark regular RRT planner with the motion primitive set pic-

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1109/ICMRE54455.2022.9734089

tured in Fig. 3 on a forest traversal problem. The benchmark-planner is a RRT algorithm employing the same motion primitive set as the RRT-Funnel algorithm, with the same LQR controller, and the same distance metric. The difference is that the benchmark planner does not take uncertainty into account, and instead maximizes the distance to the nearest obstacle as the extension operator i.e.,

$$\max_{i} \min_{t,j} \big(\mathsf{x}_i(t), o_j\big) \tag{4}$$

where $x_i(t) \in \mathcal{T}$, is a trajectory from the basic motion primitive set, and $o_j \in \mathcal{O}$ is an obstacle in the configuration space $\mathcal{C}$. Note also that in order to guide the expansion towards the goal, a goal bias of 10% is given to the benchmark planner, which is beneficial for decreasing the dispersion of the algorithm [10]. This is introduced to counter the distance metric, which tries to maximize the distance to the obstacles.

The end goal is set so that it will not take pose into account, and will only be concerned with getting within an ϵ of the (x, y) in the test map. For all the experiments an ϵ of 5m is given to the planners.

Each test-run will be run in a forest generated with a *Poisson process* method, see [14] for an introduction to Poisson processes.

The experiments records the number of collisions for each algorithm across all test-runs. The planners will run in the same environment for each test, with the same initial starting point. The environment will be redrawn using the Poisson process to generate the obstacle forest for each consecutive run. With this test setup the difference between the planners should become evident.

Before the experiments are run, all individual funnels in the base set are run with a hundred simulations runs from random starting positions in its inlet, to check if the invariant holds, and that the model stays within the funnel at all times. Uncertainty is added in terms of an additive noise with $w = 0.3$ m/s in the world x-direction, imitating a cross-wind.

A. Experiment Setup

The algorithm is tested by generating a random strip of forest of depth 25m, and then letting the RRT-Funnel, along with the benchmark algorithm find a way through the environment to the other side. Three seperate experiments are conducted where the amount of crosswind will vary. The first experiment will have a crosswind of 0 m/s, in order to set up the experiment baseline. The second experiment will have a crosswind of 0.3 m/s, which is the maximum crosswind modelled in the RRT-Funnel algorithm. This experiment will show the difference between explicitly handling uncertainty in comparison to handling it heuristically. Lastly, an experiment with a 0.6 m/s is perfomed, to verify the assumed result that if the uncertainty assumptions are broken the RRT-Funnel algorithm does no longer guarantee safe traversal.

The obstacles are imagined to be trees with trunks modelled as circles with a radius of 0.1 m, and are placed randomly on the area $O = \{\mathsf{x} \mid -50 \leq x_1 \leq 50 \text{ and } 5 \leq x_2 \leq 25\}$, as the realization of a *Poisson process* with density $\lambda = 0.2$. For each trial run, a new forest is generated in this random fashion, and the algorithms are given the task to traverse the generated map in turn.

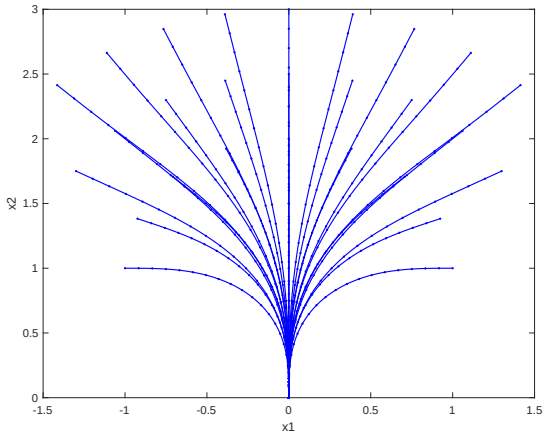

Fig. 3. The initial trajectories used in the RRT-Funnel algorithm.

The funnels for the RRT-Funnel algorithm are in this case made to handle uncertainty, in the form of a cross-wind up to and including 3 m/s in the x-direction of the airplane, but does not currently have any control over the y-direcction. Since the model used is single input, and only controls the angle of travel, not the speed of the aircraft. A maximum of 5000 nodes is set as an upper treshold on both of the algorithms' exploration trees.

B. Generating Robust Motion Primitives

This paper employs the simple unicycle model from [10] which is modified slightly into

$$\mathsf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \dot{\mathsf{x}} = \begin{bmatrix} -v\sin(\theta) \\ v\cos(\theta) \\ u \end{bmatrix}, \tag{5}$$

which is a first-order unicycle model with a constant speed. Although this is the only model used in this paper, the method can be adapted into accommodating a different and more complex model.

The trajectories for the base set of motion primitives is generated through the direct collocation method on the dynamical system given in (5).

The basis set of motion primitives should be small, yet cover enough of the finer movements of the dynamical system so that the motion of the planning unit can be near continuous when composed together. Thus in order to generate a dense set of motion primitives, points along the arc of a circle with N different radii as the initial points for the trajectory generator described in Section II-B1. The initial trajectories employed in the experiments can be seen in Fig. 3.

C. Initializing the Funnel Calculations

As noted in Section II-B2, the funnel calculations need to be initialized with a candidate Lyapunov function. For this paper, this is a TV-LQR controller.

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1109/ICMRE54455.2022.9734089

Then to get the initial Lyapunov function, the system error dynamics are linearized.

$$\dot{\bar{\mathsf{x}}} \approx A(t)\bar{\mathsf{x}}(t) + B(t)\bar{\mathsf{u}}(t) \tag{6}$$

$$\dot{\bar{\mathsf{x}}} \approx \begin{bmatrix} 0 & 0 & -v\cos(\theta) \\ 0 & 0 & -v\sin(\theta) \\ 0 & 0 & 0 \end{bmatrix} \bar{\mathsf{x}}(t) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \bar{\mathsf{u}}(t), \tag{7}$$

which is an an initial candidate Lyapunov function of the form

$$V(t, \bar{\mathsf{x}}) = \bar{\mathsf{x}}^T S_i \bar{\mathsf{x}}, \tag{8}$$

where S_i is a solution of the *Ricatti* equation

$$-\dot{S}(t) = A^T S(t) + S(t)A - \left(S(t)B + N\right) R^{-1} \left(B^T S(t) + N^T\right) + Q \tag{9}$$

associated with the LQR controller. The feedback is gained from

$$K(t) = R^{-1}\left(BS(t) + N^T\right),$$

and enables the system dynamics Eq. (5) to be written $f_{cl}(t, \mathsf{x})$ by direct substitution of $\mathsf{u} = -K\mathsf{x}$, where K is a 1×3 matrix, and hence making the system in Eq. (5) dependent only on t and x,

$$\mathsf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \dot{\mathsf{x}} = \begin{bmatrix} -v\sin(\theta) \\ v\cos(\theta) \\ -K\mathsf{x} \end{bmatrix}. \tag{10}$$

Which means that

$$V_i(t, \mathsf{x}) = \mathsf{x}^T S_i \mathsf{x}$$

as needed, where $V_i(t, \mathsf{x})$ is a quadratic Lyapunov function, which is used by the SoS optimizer to generate the funnels.

D. Generating the Funnels around the Initial Trajectories

With the inital trajectory set defined, it is time to generate the funnels around them, so that safe traversal can be guaranteed. Thus for this to happen, the funnel generator needs an initial condition set from which to start. This is given by the hyper-ellipsoid

$$\mathcal{X}_0 = \{\mathsf{x} \in \mathbb{R}^3 \mid \mathsf{x}^T Q \mathsf{x}\} \tag{11}$$

$$Q = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{bmatrix}. \tag{12}$$

which means that the dynamical system can start in a wide variety of initial states, and the controller will be able to take it safely to the outlet of the funnel. This is also beneficial, as this means it is easier to compose with another funnel, as the inlet is bigger.

E. Funnel Transformation and Invariance

Given the model from Eq. (5) the cyclic coordinates of the system are found from:

$$\begin{aligned} \mathcal{L} = T - V &= \frac{1}{2}mv^2 + \frac{1}{2}I\dot{\theta}^2 \\ &= \frac{1}{2}\left(m\left(\dot{\mathsf{x}}^2\sin^2\theta + \dot{\mathsf{x}}^2\cos^2\theta\right) + I\dot{\theta}^2\right) \\ &= \frac{1}{2}\left(m\dot{\mathsf{x}}^2 + I\dot{\theta}^2\right) \end{aligned}$$

which shows that the Lagrangian is invariant to shifts in the (x, y, θ) variables, since $\frac{\partial \mathcal{L}}{\partial q_i} = 0, q_i = x, y, \theta$. Now any funnel in the base set can be shifted freely around in the cyclic coordinates of the system without changing the solution to the system dynamic equation, and thus create an infinite set of funnels in the state space for the planner to work with. Through the partitioning of coordinates into cyclic- and non-cyclic coordinates of the form $\mathsf{x} = [x_c\ x_{nc}]^T$, the state dynamics Eq. (5) only depends on the cyclic coordinates of the system. Thus, a trajectory of the form $t \to \left(\mathsf{x}(t), \mathsf{u}(t)\right)$ which solves $\dot{\mathsf{x}} = f\left(\mathsf{x}(t), \mathsf{u}(t)\right)$ can then be transformed through a shift Ψ_c along the cyclic coordinates of the system to yield a valid solution of the form

$$t \to \left(\Psi_x(\mathsf{x}(t)), \mathsf{u}(t)\right)$$

where the transform Ψ is given by

$$\Psi\left(\begin{bmatrix} \mathsf{x}_c \\ \mathsf{x}_{nc} \end{bmatrix}\right) = \begin{cases} \mathsf{x}_c \to \mathsf{x}_c' \\ \mathsf{x}_{nc} \to \mathsf{x}_{nc}. \end{cases}$$

However, since $\dim(\mathsf{x}) = \dim(\mathsf{x}_c)$, for the dynamics equation in 5, it is not necessary to handle the non-cyclic case for the model in this experiment.

F. Funnel Composition

The funnel robustness guarantees are only valid if the funnels are composable. Unfortunately, the funnels do not compose, and the off-line composition testing of the algorithm has to be left out. Thus the experiments are run with a funnel graph that is complete, and all funnels are checked on-line if they compose with each other, at the cost of on-line complexity during the algorithm's execution. This is because the one dimensional controller employed has no influence on the speed of the airplane, and hence there is no way to make the system converge in the direction of speed. This is exemplified in Fig. 4, where the inlet is overlaid the outlets for the projected xy-funnel, and it can be seen that the controller is able to converge the xy-funnel in the x-direction, but not in the y-direction, as it has no control over this dimension at all. The framework can be expanded with this functionality. however, this is referred to as future work.

In order to remedy this off-line compositional robustness guarantees, the RRT-Funnel algorithm keeps track of whether the model has left the funnel during execution, and aborts the simulation with the emergency maneuver if the airplane leaves one of the funnels at runtime. This happens if the value of the Lyapunov function is larger than one. In the experiments, this counts as a collision on the part of the RRT-Funnel algorithm.

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1109/ICMRE54455.2022.9734089

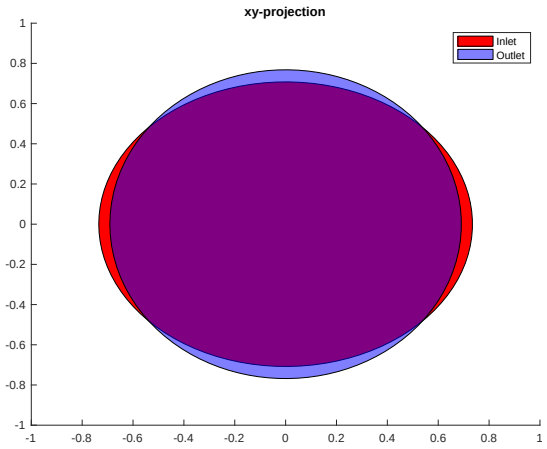

Fig. 4. The projection of the funnel inlet and outlet onto the xy-plane, in meters. It is seen that the controller is able to make the funnel converge in the x-direction as expected, however, it has no control in the y-direction, as there is no controller steering the speed of the vehicle.

G. The Size of the Airplane and the Obstacles Models

The funnels generated thus far are created from a point model of the airplane, and its dynamics. If the grid that the simulations are run on are set to have an unit size of one meter, then the funnels from the basic set are given a velocity of $[v(t)] = \text{m/s}$, $[\theta] = \text{rad}$, and $[\dot{\theta}] = \text{rad/s}$, where $[\,\cdot\,]$ is the unit operator. In this experiment we assume that the aircraft if a small radio controlled aircraft with a maximal speed of 10 m/s and a size of 10×20 cm, however the speed and size of the aircraft can be set arbitrary. The mass is not relevant for our first order dynamics, but still the airplane is assigned a mass of $1\,kilogram$, so that the translation of the model dynamics is not irrelevant.

H. Expanding the Funnels around the Airplane Model

The size of the airplane in the original model is a single point, and as such, the expanded airplane model is not accounted for in the current funnels. Therefore the funnels have to be expanded in order for them to accommodate the airplane model. However, the size of the airplane only affects the size of the funnel ellipsis projected down into the xy-plane. Therefore, first extract the projected size of the funnel, through a projection map: $P\colon \mathbb{R}^3 \to \mathbb{R}^2$, where $P = \begin{bmatrix} I_{2\times 2} & 0_{2\times 2} \end{bmatrix}$ such that for the projected ellipsoid $\mathcal{E}_p = \{\bar{\mathsf{x}} \in \mathbb{R}^2 \mid \bar{\mathsf{x}}^T S_k^{(p)} \bar{\mathsf{x}} \leq 1\}$, with $S_k^{(p)}$ given by $S_k^{(p)} = \left(P S_k^{-1} P^T\right)^{-1}$, is the set containing the funnel projected down into the xy-plane. Here $\mathcal{E}_p$ is the projected set of the ellipsoid in the xy-plane. In general an ellipse centered at the origin is a linear transformation of the unit circle [15]. Exploiting this fact, the funnel ellipsoids can be expanded to encompass the airplane model. Take note that the matrix S_k is *Positive semidefinite*, and hence can be Cholezky factorized [15]. The expanded ellipsis (which now contains all the possible states of the airplane model) is given by:

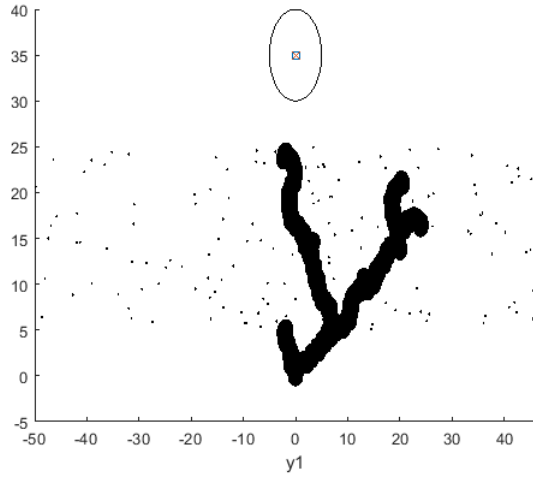

Fig. 5. Visualized is the expansion of the RRT-Funnel algorithm at 101 iterations of the algorithm with the funnels shown. Both axis are in meters.

$$
\begin{aligned}
S_k^{(\mathcal{P})} &= R^T R \\
\mathsf{x} &= R^{-1}\mathsf{y} \\
\mathcal{C} &= \{\mathsf{y} \in \mathbb{R}^2 \mid \mathsf{y}^T\mathsf{y} \leq 1 + r_a\} \\
\mathcal{E}_{exp} &= \{R^{-1}\mathsf{y} \mid \mathsf{y} \in \mathcal{C}\}
\end{aligned}
$$

where $\mathcal{E}_{exp}$ are the ellipsoids which contains the volume of the airplane for all verified states in the funnel, and r_a is the widest part of the model at hand, which in this case is the wingspan. A picture of the initial funnel and the funnel expanded around the airplane model can be seen in Fig. 2.

IV. Results

The results from the given simulations are summarized through the number of collisions, the total length of the solution trajectories, and the number of branches added to each search tree. These performance measures are shown in Table I, Table II and Table III respectivly. For each column total of hundred simulation is performed. In the first table a count of how many of the simulations runs that resulted in a collision is given, while in the latter two tables an average of the hundred simulation runs are given. Visualizations of the simulations can be seen in Fig. 5 where the funnels are shown, and in Fig. 6 where only the search tree is shown. A plot of the Lyapunov values for a simulation run can be seen in Fig. 7.

V. Discussion

Currently the algorithm handles uncertainties in position, but not in the environment, and must therefore be run in a known environment, and is hence a strictly off-line motion

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1109/ICMRE54455.2022.9734089

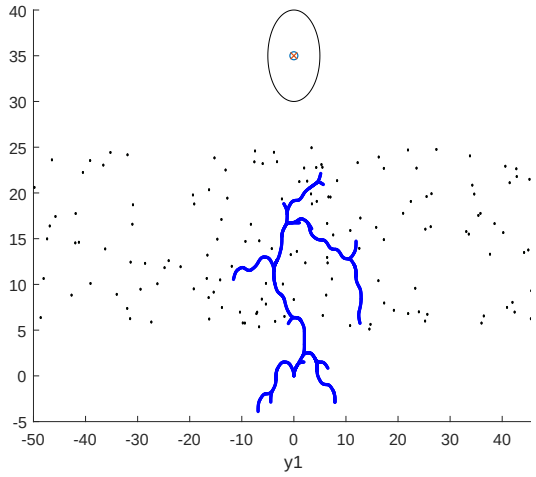

Fig. 6. Visualized is the expansion of the RRT-Funnel algorithm at 101 iterations of the algorithm with the tree shown. Both axis are in meters.

TABLE I. THE TOTAL NUMBER OF COLLISIONS FOR EACH ALGORITHM OVER A TOTAL OF 100 SIMULATION RUNS, WITH THREE DIFFERENT VALUES FOR THE CROSS-WIND (W).

Number of Collisions	$w = 0\,m/s$	$w = 3\,m/s$	$w = 6\,m/s$
RRT − Funnel	0	0	4
Benchmark	0	6	10

TABLE II. THE TOTAL LENGTH OF THE SOLUTIONS FOUND FOR EACH ALGORITHM OVER A TOTAL OF 100 SIMULATION RUNS, WITH THREE DIFFERENT VALUES FOR THE CROSS-WIND (W).

Solution Trajectory's Length	$w = 0\,m/s$	$w = 3\,m/s$	$w = 6\,m/s$
RRT − Funnel	1.506	3.418	4.754
Benchmark	3.582	4.162	3.174

TABLE III. THE TOTAL NUMBER OF ITERATIONS FOR EACH ALGORITHM OVER A TOTAL OF 100 SIMULATION RUNS, WITH THREE DIFFERENT VALUES FOR THE CROSS-WIND (W).

Number of Iterations	$w = 0\,m/s$	$w = 3\,m/s$	$w = 6\,m/s$
RRT − Funnel	447.646	190.713	213.638
Benchmark	29.120	52.768	33.936

planner. However, it is possible to generalize the algorithm to handle unknown environments.

In general, the funnels generated are tight outer approximations of the true reachable set for the system. However, note that since the Lyapunov function employed is quadratic, it will always be symmetric around the trajector verified. This means that even though the real nonlinear system dynamics can have a tight reachable set on one side of the trajector, the symmetry of the quadratic Lyapunov function might lead

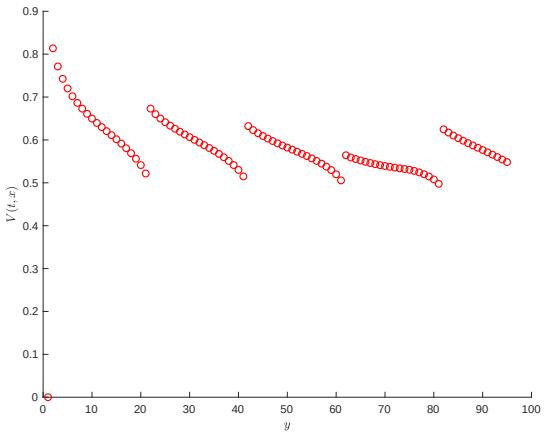

Fig. 7. The plot of the Lyapunov values for a simulation run at the sampling times t_k.

the funnel to be too conservative on one side of the trajector. For this paper however, the system dynamics are symmetrical, so this is not an issue. That the funnels generated do provide tight outer approximations was verified through a Monte-Carlo simulation of the nonlinear system with bounded uncertainty. Over the course of 100 simulations the system never left any of the funnels. Thus showing that the funnels did provide tight outer approximations of the dynamics of the system.

The strength of the algorithm lies in that it can separate handling the uncertainty into an off-line pre-computation phase. Therefore, the global motion planner does not need to be significantly more complex than if it had not taken uncertainty into account. In fact, it can remain oblivious to the overarching problem difficulty of handling uncertainties for a complex nonlinear system. In fact, once the motion primitives have been calculated and verified off-line, they might as well be employed in any global motion planner able to handle discrete motion primitives.

Even though the robustness guarantees of the SOS framework could not be guaranteed in the off-line phase, due to the planner not handling multiple controller inputs, and hence the funnels did not compose off-line. Still, the RRT-Funnel algorithm did run-time verification of funnel abidance. Even though the model leaving a non-composable funnel at run-time was theoretically possible, this did not cause the airplane to collide a single time over the course of 300 simulation runs. This was in starch contrast to the benchmark planner, which did not handle uncertainty at all, and instead relied on avoiding the obstacles by as big a margin as possible, and therefore consistently had a collision rate of 6% or higher. This collission rate can be expected to be a lot higher in a denser planning environment, but this would also significantly add to the time of the off-line planning phase. As can be seen in the last column, when the cross-wind added to the experiment violated the upper bound of 3 m/s, the RRT-Funnel algorithm did also start crashing. It is seen that the RRT-Funnel algorithm performs better in terms of robustness up to and including its uncertainty bound of $w = 3$ m/s, while the benchmark

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1109/ICMRE54455.2022.9734089

planner does collide in the same environment, with the same uncertainty.

Although the RRT-Funnel algorithm performed better in terms of collisions, it does a lot worse in terms of exploring the planning space, than does the benchmark planner. This only increases with the difficulty of the planning space, and hence the time spent by the RRT-Funnel algorithm is significantly longer.

VI. Conclusion

This paper has shown that the RRT-Funnel motion planning algorithm does provide robust feedback motion planning for a nonlinear dynamic system. It shows that a motion planning algorithm employing discrete verified robust motion primitives outperforms a traditional motion planning algorithm significantly in terms of safe traversal through a known environment. It has also shown that the robust planner is only reliable up to and including its uncertainty bounds, and will start misbehaving, just like the benchmark motion planner, once the uncertainty assumptions on the algorithm are broken. All in all, it is shown that the RRT-Funnel algorithm is a viable option for a global motion planner in an uncertain environment.

References

[1] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: A survey," vol. 33, no. 03, pp. 463–497. [Online]. Available: http://www.journals.cambridge.org/abstract_S0263574714000289

[2] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone, "Robust online motion planning via contraction theory and convex optimization," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5883–5890. [Online]. Available: http://ieeexplore.ieee.org/document/7989693/

[3] R. Tedrake, "LQR-trees: Feedback motion planning on sparse randomized trees," in *Robotics: Science and Systems V*. Robotics: Science and Systems Foundation. [Online]. Available: http://www.roboticsproceedings.org/rss05/p3.pdf

[4] B. D. Luders, S. Karaman, and J. P. How, "Robust sampling-based motion planning with asymptotic optimality guarantees," in *AIAA Guidance, Navigation, and Control (GNC) Conference*, 2013, p. 5097.

[5] N. A. Melchior and R. Simmons, "Particle rrt for path planning with uncertainty," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 1617–1624.

[6] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," vol. 36, no. 8, pp. 947–982. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364917712421

[7] M. M. Tobenkin, I. R. Manchester, and R. Tedrake, "Invariant Funnels around Trajectories using Sum-of-Squares Programming," vol. 44, no. 1, pp. 9218–9223. [Online]. Available: https://doi.org/10.3182%2F20110828-6-it-1002.03098

[8] A. Majumdar and R. Tedrake, "Robust Online Motion Planning with Regions of Finite Time Invariance," in *Algorithmic Foundations of Robotics X*, E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus, Eds. Springer Berlin Heidelberg, vol. 86, pp. 543–558. [Online]. Available: http://link.springer.com/10.1007/978-3-642-36279-8_33

[9] A. A. Ahmadi and A. Majumdar, "DSOS and SDSOS optimization: LP and SOCP-based alternatives to sum of squares optimization," in *2014 48th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, pp. 1–5.

[10] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, available at http://planning.cs.uiuc.edu/.

[11] J. T. Betts, "Survey of Numerical Methods for Trajectory Optimization," vol. 21, no. 2, pp. 193–207. [Online]. Available: https://doi.org/10.2514%2F2.4231

[12] O. Von Stryk, "Numerical solution of optimal control problems by direct collocation," in *Optimal Control*. Springer, pp. 129–143.

[13] J. Kuffner, "Effective sampling and distance metrics for 3D rigid body path planning," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*. IEEE, pp. 3993–3998 Vol.4. [Online]. Available: http://ieeexplore.ieee.org/document/1308895/

[14] D. P. Kroese and Z. I. Botev, "Spatial Process Generation," p. 41.

[15] D. C. Lay, "Linear Algebra and its Applications, 3rd updated Edition."