AutoPark: Path Finding with RRT*

Anton Lukyanenko, Heath Camphire, Damoon Soudbakhsh,
Avery Austin, Carlos Guerra, Samuel Schmidgall

Mason Experimental Geometry Lab

May 3rd, 2019

History

This project was started last summer by Dr.Lukyanenko and Dr.Soudbakhsh with a simple conversation on how math and controls intersect.

Champions

- Heath Camphire
- Abigail Shoemaker
- Jiwei Qin
- Sanjida Nasreen

Introduction

Self-Driving cars are quickly being integrated into society, but there are still lacking any motion-planning algorithms that effectively find paths for multiple cars.

Problem

If we have a self-driving car filled parking garage and we want to get one of the cars out, how can we maneuver the cars around it so that the car in question leaves the garage?

Solution

Build an efficient algorithm that optimally maneuvers multiple cars into specified positions.

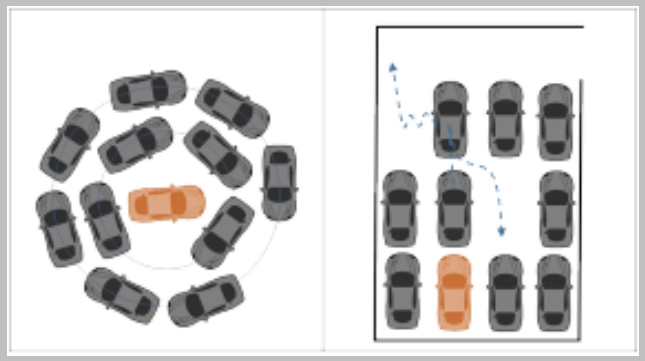


Figure 1: Parking configurations that require cooperative motions for the subject car to leave

What is an optimal path, anyways?

Since cars are constrained by a turning radius, we cannot use Euclidean distance to measure path length. So, what do we use?

This problem was solved in 1990 by Reeds and Shepp. They concluded that there are 48 unique motion sequences that guarantee your path will be optimal. The motion sequences are constructed from up to 48 different combinations of motion primitives (left curve, right curve ...)

Base	α	β	γ	d
$C_\alpha \vert C_\beta \vert C_\gamma$	$[0, \pi]$	$[0, \pi]$	$[0, \pi]$	–
$C_\alpha \vert C_\beta C_\gamma$	$[0, \beta]$	$[0, \pi/2]$	$[0, \beta]$	–
$C_\alpha C_\beta \vert C_\gamma$	$[0, \beta]$	$[0, \pi/2]$	$[0, \beta]$	–
$C_\alpha S_d C_\gamma$	$[0, \pi/2]$	-	$[0, \pi/2]$	$(0, \infty)$
$C_\alpha C_\beta \vert C_\beta C_\gamma$	$[0, \beta]$	$[0, \pi/2]$	$[0, \beta]$	–
$C_\alpha \vert C_\beta C_\beta \vert C_\gamma$	$[0, \beta]$	$[0, \pi/2]$	$[0, \beta]$	–
$C_\alpha \vert C_{\pi/2} S_d C_{\pi/2} \vert C_\gamma$	$[0, \pi/2]$	-	$[0, \pi/2]$	$(0, \infty)$
$C_\alpha \vert C_{\pi/2} S_d C_\gamma$	$[0, \pi/2]$	-	$[0, \pi/2]$	$(0, \infty)$
$C_\alpha S_d C_{\pi/2} \vert C_\gamma$	$[0, \pi/2]$	-	$[0, \pi/2]$	$(0, \infty)$

Figure 2: 48 Reeds-Shepp motion primitive combinations

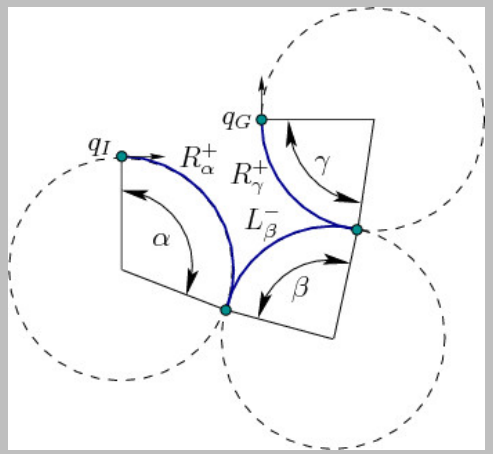


Figure 3: Optimal Trajectories in Reeds-Shepp Space

Obstacles are in the way of optimal paths

The catch is that this only works in obstacle free environments.

Well, we still want to guarantee optimality, but in obstacle environments the Reeds Shepp paths no longer generate optimal trajectories

To get around this, we introduce an algorithms that maintains a tree of trajectories that are appended together in a manner which allows for us to eventually generate an optimal path.

Rapidly-Exploring Random-Tree*

To predict motion for a single car, the algorithm RRT* is commonly used. Our team chose to use RRT* for the path-finding aspect of our project.

Overview of RRT*

RRT*, which is short for Rapidly-Exploring Random-Tree*, is the current state-of-the-art path-finding algorithm that iteratively explores a state space.

Why RRT*?

- RRT* is unique in that it converges to the optimal solution.
- RRT* allows for differential constraints – such as a turning radius.

RRT* Algorithm

First, a random sample is generated from the state space. In a one-car environment this is usually an n-dimensional vector with n-m components corresponding to rotation and the other m corresponding to position.

Next, the nearest neighbor for that sample is found. If the path connecting the two is obstacle free then it is added to the tree. Finally, the key aspect that allows for the algorithm to converge to the optimal trajectory is we check if the newly added node makes a more optimal parent for any of the nodes nearby.

Rapidly-Exploring Random-Tree*

Multiple Car RRT* Algorithm

In our case, RRT* was not a sufficient because it only considered the path of one car. So, our team extended the original RRT* algorithm to work with multiple cars by maintaining a tree where the nodes correspond to multiple car's positons and the trajectories are for each car respectively.

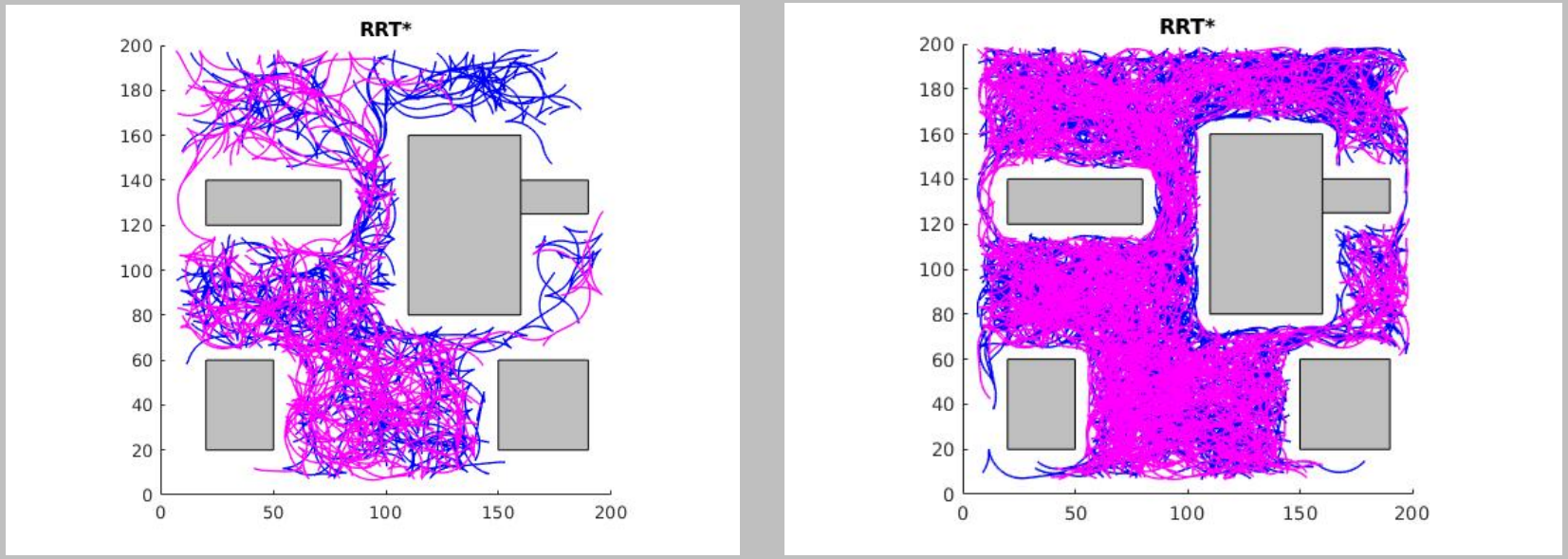
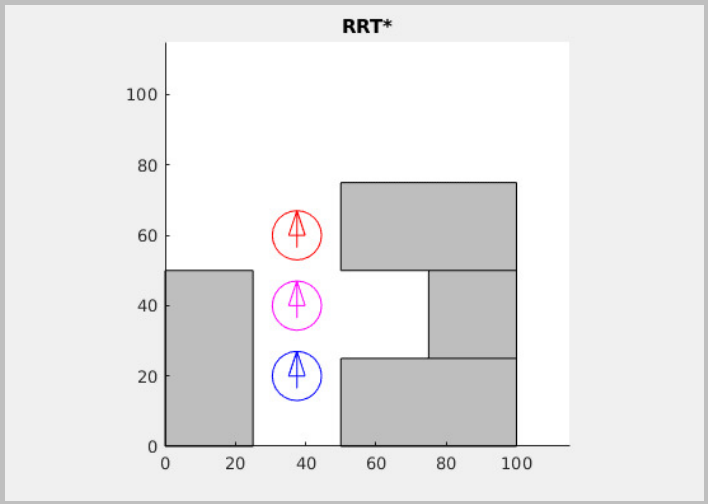


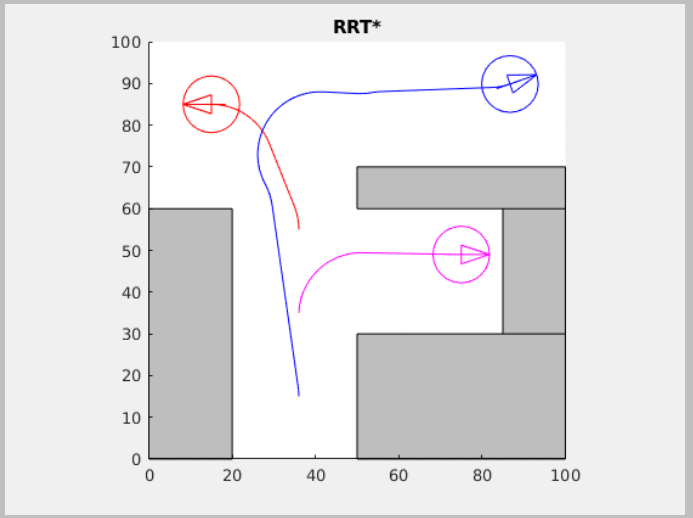
Figure 4: Reeds-Shepp RRT* Tree Visualization with 500 and 1500 nodes respectively

Simulation

A significant portion of our effort has been on developing simulations for multi-car RRT*.

Originally everything was done in MATLAB. This code was slow and did not develop good paths. Since then we have rewritten all of the code in C++ and reduced the computation time significantly.





Prime Modulus Sampling

We also proposed a sampling procedure which has empirically shown to generate better paths in shorter time.

$ModulusArray \leftarrow [Prime_1, Prime_2, ...Prime_n]$
Such that n = number of cars and $Prime_m$ is unique
for $i \leftarrow 0$ **to** *iterations* **do**
  positionSample $\leftarrow$ []
  for $j \leftarrow 0$ **to** *numCars* **do**
    if i *mod ModulusArray[j]* $== 0$ **then**
      positionSample[j] = ($goal_j$X, $goal_j$Y, $goal_j\theta$)
    else
      positionSample[j] = (randomX, randomY, randomθ)
  end
  RRT*-Stuff(positionSample)
end

Algorithm 1: Prime Modulus Sampling

Implementation Goal

The idea then is to take the simulation data and experiment its effectiveness in a pseudo real world environment. How is this done?

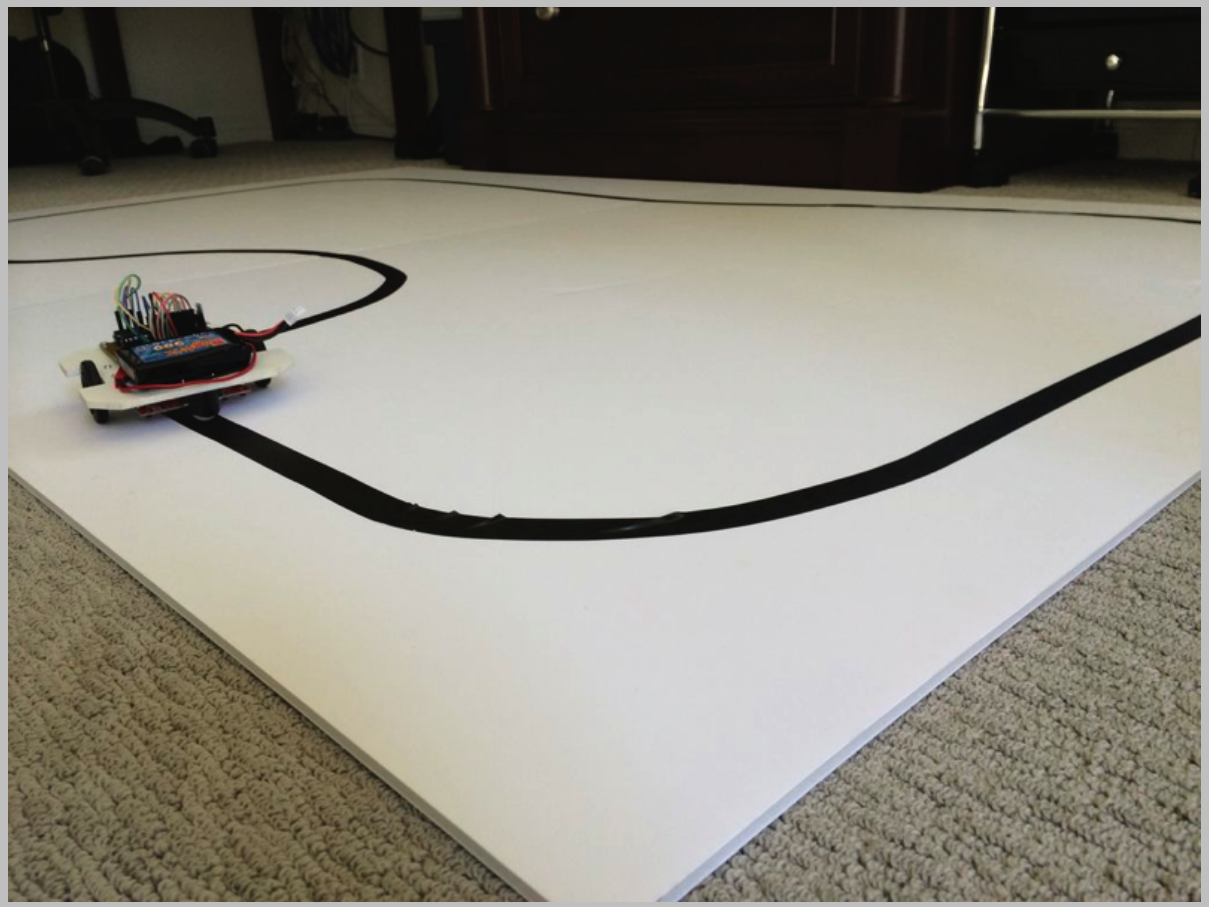
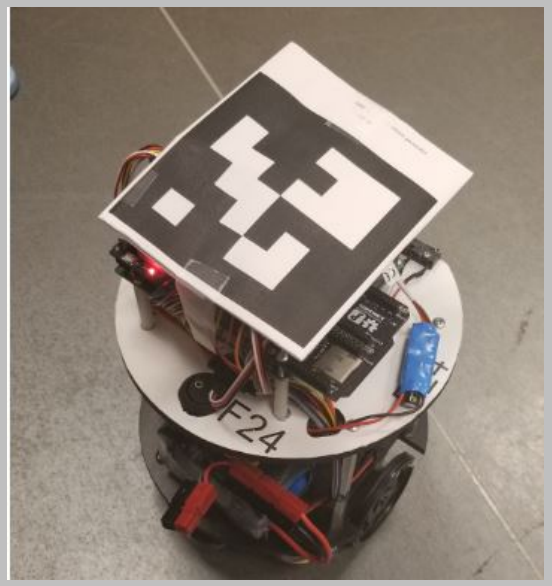


Figure 5: Random robot following a line.

FlockBot Implementation



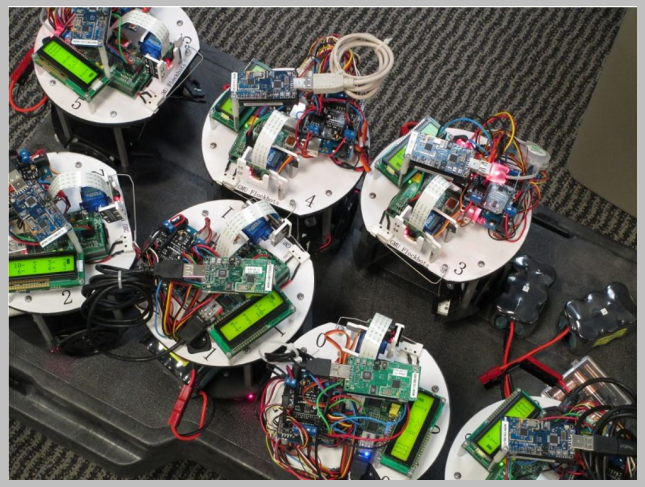


Figure 6: We observe multiple AR tags using a camera. Then each robot is sent individual commands based on its location.

Controlling the Flockbots

Closed Loop Feedback Control System

Used to determine the left and right wheel velocities, V_L and V_R, given it's axel length, L.

$$V = (V_R + V_L)/2$$
$$\omega = (V_R - V_L)/L$$
$$R = L(V_L + V_R)/2(V_R - V_L)$$

PID Error Correction

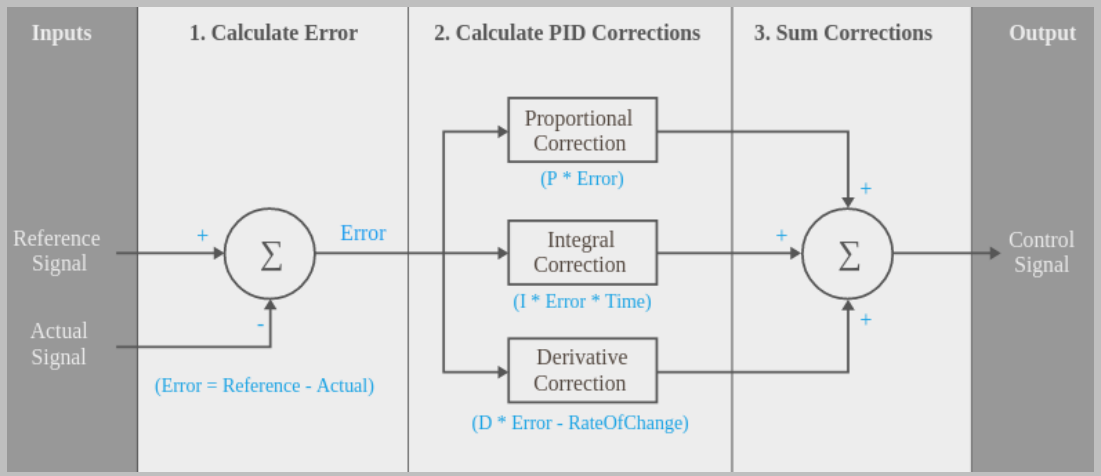


Figure 7: A high-level overview of a PID Controller

Flockbot Trust Issues

We encountered many problems throughout the semester.

- Contrast
- Gain
- Brightness
- PID tuning
- Camera dropping frames
- Wheels slipping
- Contrast
- Gain
- Brightness
- PID tuning
- Camera dropping frames
- Wheels slipping
- Saturation
- Elongated camera coordinates
- TCP connections
- Floor reflectively
- Error tolerances
- Goal error tolerances

The most important issue being the fidelity of the Flockbots. Initially, we could only tell if a Flockbot was on the correct path by human sight.

Flockbot Trust Issues (cont.)

We fixed this issue by coloring pixels on the screen. We observed the pixel coordinates of the 4 corners of the AR tags. We then took the average to determine the center and changed its color.

Figure 8: Flockbots following a figure 8 (left) and a circle path (right)

Play Video

Future Goals

Real-Time

In addition to the traditional simulation, we have started developing code for RRT* to be run in real-time. This will allow for moving obstacles and second to second tree adjustments. This effort would involve:

- Developing an effective tree pruning algorithm
- Dynamically detecting and handling obstacles coming in and out of frame.
- Integrating the two code bases for simulation and Flockbot control.