Project Report - Implementation of the algorithms from a paper (A Distributed Feedback Motion Planning Protocol for Multiple Unicycle Agents of Different Classes by Dimitra Panagou)

Oluwadamilola Osayomi - 1003648617
University of Toronto Institute for Aerospace Studies

April 24, 2018

Abstract

The paper in focus for this project presents a way of creating a safe environment for multiple unicycle objects moving around in the same space, without being in the same class. The term class here is used to differentiate object can communicate with each other and those that can't.

As explained in the paper, the agents in class A are able to communicate their position and orientation with other agents of class A. The objects in class B, however, do not communicate this information with the agents of class A. Although, class A agents can get this information by sensing if they are close enough. The path planning and obstacle avoidance is based on a family of 2-D vector fields. This idea was successfully implemented for a single class A agent case and scaled up to multiple class A agents using a semi-cooperative coordination concept that provides conflict resolution between these agents by adjusting their speeds individually.

The problem being solved: Multi-agent path planning and coordination issues in compelled dynamic conditions (i.e. conditions where operators have limited detecting and communication).

Contents

List of Figures

1 Introduction

This report explains an attempt to implement the studies from a paper (Panagou, 2014a). The reason behind this project is to better understand the logic and algorithms used in the research and possibly identify areas of improvement on the authors work. Although this paper was the sole focus of the project, insight was drawn from other sources as well. The paper introduces a path planning concept that creates a safe path for multiple vehicles (all unicycle model in this case) using 2-D vector fields. The concept utilizes attractive fields that always almost converge to the goal position and repulsive fields that flow around obstacles. The concept was successfully implemented for the single agent case(class A), however, scaling up to multiple class A agent requires cooperative coordination between these agents which is beyond the scope of this project. The model used in this project is a unicycle model based on what was used by (Panagou, 2014a) as analytically shown below. It is also important to note that the paper being investigated does not cover the effect of disturbances such as measurement noise, but these are covered in more detail later in (Garg & Panagou, 2018)

For the sake of this report, the class A agents will be referred to as vehicles and the class B agents referred as obstacles from now on.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ \omega \end{bmatrix} \tag{1}$$

where $\dot{x}, \dot{y}, \dot{\theta}$ represent the vehicle 2D motion and u and ω are the linear and angular velocity of the robot. The vehicle and obstacles are modeled as 2-D closed circular objects The safe path is generated from the equation below and is the basis of the entire concept, which is a family of 2-D vector fields also obtained from (Panagou, 2014a).

$$F(r) = \lambda(p^T r)r - p(r^T r), \tag{2}$$

where $\lambda \in \Re$ determines the type of vector field, $p = \begin{bmatrix} p_x & p_y \end{bmatrix}^T$ represent the field orientation and $r = \begin{bmatrix} x & y \end{bmatrix}^T$ is the position vector w.r.t the cartesian frame.

Simplifying equation (2), we have:

$$F_x = (\lambda - 1)p_x x^2 + \lambda p_y xy - p_x y^2, \tag{3a}$$

$$F_y = (\lambda - 1)p_y y^2 + \lambda p_x xy - p_y x^2, \tag{3b}$$

2 Algorithms involved

The motion planning and obstacle avoidance are based on vector fields. In simple terms. The object travels to its target location through a vector field that

is attractive to and guarantees almost global convergence to its goal position r_g, and is able to avoid obstacles through one that is repulsive around the obstacles as illustrated in the figures (1) & (2) & (3) below. These figures were obtained using an in-built matlab function called 'streamslice' that takes in an array of x, y values along with their corresponding F_x, F_y obtained from eqn (3) and outputs a plot showing what the field looks like. This is particularly helpful in visualizing the expected path of the vehicle to its goal position r_g. **To accomplish the implementation of this algorithm, Matlab software was used.**

2.1 Attractive field

Substituting $\lambda = 2$, $p_x = 1$ & $p_y = 0$ into eqn(3), one gets the equation that governs (1).

$F_x = x^2 - y^2$

$F_y = 2xy$

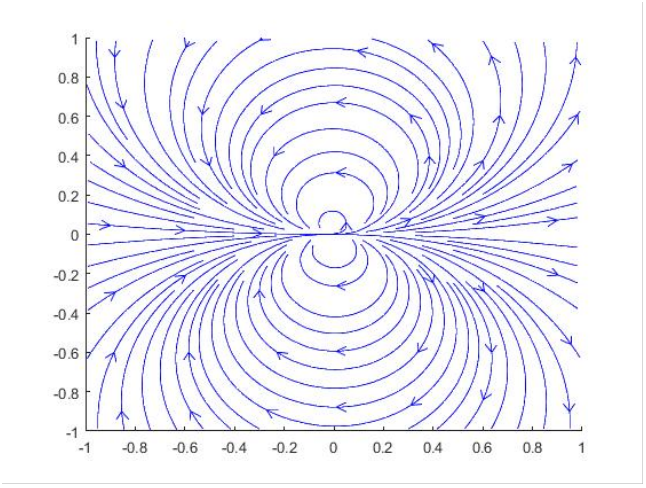

Figure 1: *Integral curves of eqn(1) for* $\lambda = 2$, $p_x = 1$, $p_y = 0$.

2.2 Repulsive field

Substituting $\lambda = 1$, $p_x = 1$ & $p_y = 0$ into eqn(3), one gets the equation that governs (2).

$F_x = -y^2$

$F_y = xy$

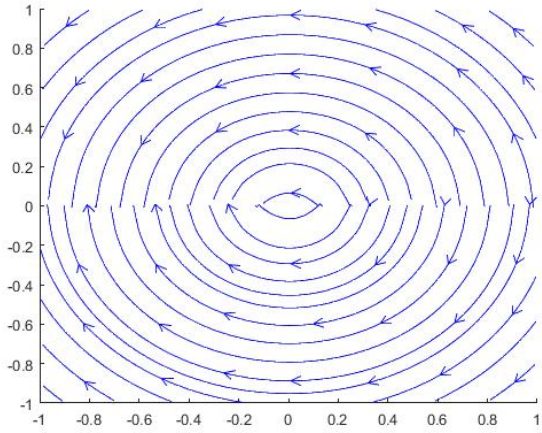

Figure 2: *Integral curves of eqn(1) for* $\lambda = 1,\ \ p_x = 1,\ \ p_y = 0.$

2.3 Parallel-to-goal field

Substituting $\lambda = 1,\ p_x = 1\ \ \&\ p_y = 0$ into eqn(3), one gets the equation that governs 3.

$F_x = -x^2 - y^2$

$F_y = 0$

This is particularly useful to help the vehicle escape the field around obstacles, this idea will be further explained in the coming sections.

3 Implementation Part I

A Matlab code was generated to implement the algorithms presented by (Panagou, 2014a). The goal was to successfully implement the case for a single vehicle with multiple obstacles. An overview of the Matlab implementation has been provided below in fig (4) and each section of the code will be further explained. The vehicles and obstacles are defined as closed 2-D circular objects with radii ρ & ρ_o respectively. The obstacles were also defined to be a minimum distance of ρ_f apart to avoid conflict between repulsive fields, as this could lead to unwanted results.

3.1 Attract_functn

Obtained from equation (3), this generates a vector field that always leads to the goal position $r_g = [x_g, y_g]^T$ can be written as below, where $\lambda = 2,\ p_x = 1,\ p_y = 0\ \&\ r = [x\ \ y]^T$ from eqn(2)is taken as $r - r_g$, which yields $F_x = (x - x_g)^2 - (y - y_g)^2\ \ \&\ \ F_y = 2(x - x_g)(y - y_g)$ and fig (1)

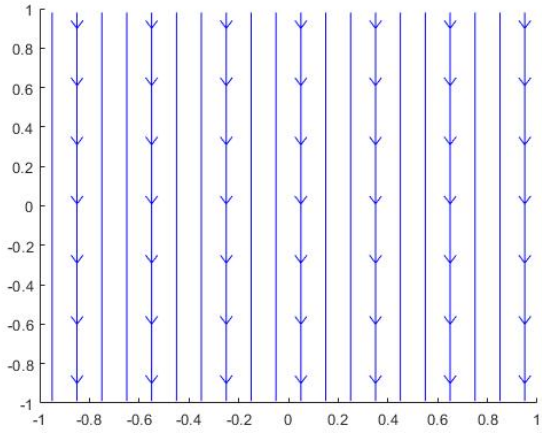

Figure 3: *Integral curves of eqn(1) for* $\lambda = 0, \quad p_x = 1, \quad p_y = 0$

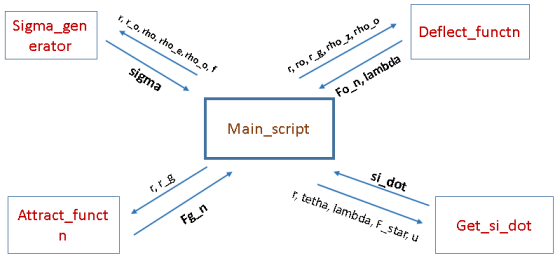

Figure 4: *A simple flow diagram showing how the matlab code layout for a single vehicle-multiple obstacle case.*

$$F_g^n = \begin{cases} \frac{F_g}{\|F_g\|} & \text{for } r \neq 0 \\ 0 & \text{for } r = 0 \end{cases} \tag{4}$$

This function basically takes in the object location & goal location (r, r_g) from the main_script script and returns the normalized attractive vector F_g^n, which by definition is zero(0) once the vehicle reaches the goal position. This is however never the case, as F_g approaches but is never equal to zero along the path to r_g.

3.2 Deflect_functn

To create the repulsive field around every obstacle at position $r_{oi} = [x_{oi}\ , y_{oi}]^T$. We can obtain a repulsive field F_o^n from eqn(3) as:

$$\begin{aligned} F_{oxi} &= p_{yi}(x - x_{oi})(y - y_{oi}) - p_{xi}(y - y_{oi})^2, \\ F_{oyi} &= p_{xi}(x - x_{oi})(y - y_{oi}) - p_{yi}(x - x_{oi})^2, \end{aligned} \tag{5}$$

When $\lambda = 1$

&

$$\begin{aligned} F_{oxi} &= -p_{xi}(x - x_{oi})^2 - p_{xi}(y - y_{oi})^2, \\ F_{oyi} &= -p_{yi}(x - x_{oi})^2 - p_{yi}(y - y_{oi})^2, \end{aligned} \tag{6}$$

When $\lambda = 0$

These equations then gives birth to a field that flows around every obstacle along the predefined repulsive field and also offers an escape route out of this repulsive field as described by eqn(7) & shown in fig(5):

$$F_g^n = \begin{cases} \frac{F_{(\lambda=0)}(\delta r_i)}{\|F_{(\lambda=0)}(\delta r_i)\|}, & \text{if } {p_i}^T(\delta r_i) < 0 \\ \frac{F_{(\lambda=1)}(\delta r_i)}{\|F_{(\lambda=1)}(\delta r_i)\|}, & \text{if } {p_i}^T(\delta r_i) \geq 0, r \notin \nu_i \\ 0, & \text{if } {p_i}^T(\delta r_i) \geq 0, r \in \nu_i \end{cases} \tag{7}$$

where $p = [p_x\ ,\ p_y]$, $\lambda = \ 1,\ \phi = \ \arctan 2\left(-y_o,\ -x_o\right),\ p_x \ = \ \cos\left(\phi\right),\ p_y \ = \ \sin\left(\phi\right)$ & $\delta r_i \ = \ r \ - \ r_{oi}$

Fig (5) provides a visual representation of the repulsive field around every obstacle. AS defined earlier by eqn(7), the red arrows are operating under eqn(5) as $\lambda = 1$ in region A_i and the green arrows under eqn(6) as $\lambda = 0$ in region B_i. Therefore by definition, the field in this region is always parallel to a line drawn from the center of the obstacle(r_{oi}) to the goal position, since vector p_i lies on that line by definition. The reason for defining a separate field for region B_i is to prevent the vehicle from getting trapped in the repulsive field in eqn(5)

This function takes in the location of the vehicle, obstacle(i) and goal location($r,\ r_{oi},\ r_g$) and outputs a normalized repulsive vector F_o along with the value of λ for each obstacle with respect to that obstacle (to be used to calculate $\dot{\varphi}$ later).

3.3 Sigma_generator

Now that we have defined the both the attractive and repulsive paths, we now need to blend them together smoothly. To do this, a variable sigma is used (where $0 \leq \sigma \leq 1$). Below is how the the algorithms were blended.

$$F^\star \ = \prod_{i=1}^{N} \sigma_i F_g \ + \ \sum_{i=1}^{N} \left(1 - \sigma_i\right) F_{oi} \tag{8}$$

Sigma is obtained using the algorithm below

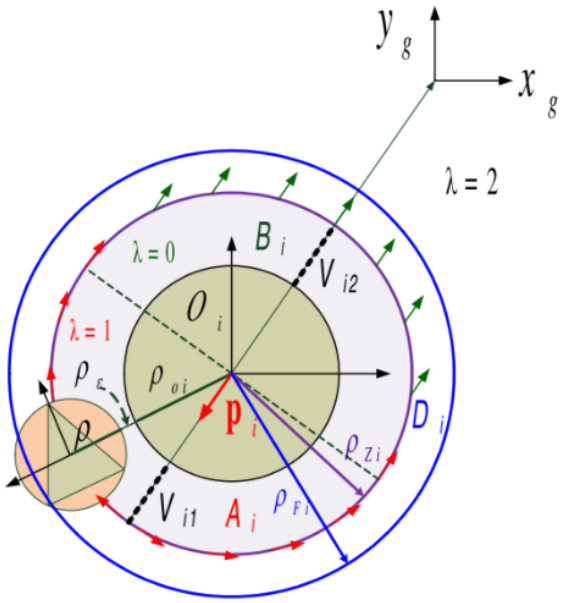

Figure 5: The repulsive field around each obstacle where $\lambda = 1$ for region A_i and $\lambda = 0$ for region B_i (panagou2014)

$$\sigma = \begin{cases} 1, & \text{for } \beta_i(r) < \beta_{Fi} \\ a{\beta_i}^3 + b{\beta_i}^2 + c\beta_i + d, & \text{for } \beta_{Fi} \leq \beta_i(r) \leq \beta_{Zi} \\ 0, & \text{if } \beta_{Zi} < \beta_i(r) \end{cases} \tag{9}$$

The equations (6) & (7) above makes sure that the vector field operates as thus:

- It is only attractive to the goal position when $\sigma = 1$ and this occurs when the vehicle is not in the boundary F defined around each obstacle I.e. the blue circle in (5)
- It is only repulsive around each obstacle when $\sigma = 0$ and this occurs when the vehicle is in the boundary Z defined around each obstacle I.e. the purple circle in (5)
- It has a blend of both attractive and repulsive fields when $0 < \sigma < 1$ and this occurs when the vehicle is in between region F and region Z

This function basically takes the location of the vehicle and each obstacle along with their radii and a distance f from the region Z (to define the region F). The region Z has a radius that is the sum of the radius of the vehicle ρ and the obstacle ρ_o and a minimum distance allowed between them ρ_ε as its inputs

and returns a value of σ for each obstacle, that is then stored in an array for later use. This is done at every iteration of the code.

3.4 Get_si_dot

The time derivative part of equation 9(b) is shown below.

$$\dot{\varphi} = \left(\left(\frac{\partial {F_y}^\star}{\partial x} c\theta + \frac{\partial {F_y}^\star}{\partial y} s\theta \right) {F_x}^\star - \left(\frac{\partial {F_x}^\star}{\partial x} c\theta + \frac{\partial {F_x}^\star}{\partial y} s\theta \right) {F_y}^\star \right) u \tag{10}$$

This section of the code takes in the object location, its orientation, the value of λ obtained from the λ array which is one of the outputs of the deflect_function (since the value for λ for each obstacle is dependent on the position of the vehicle at every iteration & is singular to each obstacle), the value of the vector $F^\star$ from eqn(6) at that location, and the linear velocity u of the object. The output is simply $\dot{\varphi}$ which is added to the angular velocity term.

3.5 Main_script

A safe distance was kept between the obstacles themselves, in order to avoid conflicting vector fields, so that the vehicle is never stuck in two repulsive regions at the same time as this is an unwanted occurrence.

$$u = k_u \tanh(\|r - r_g\|) \tag{11a}$$
$$w = -k_\omega(\theta - \varphi) + \dot{\varphi} \tag{11b}$$

Eqn (9) defines the control design for the vehicle described in eqn(1) Where,

- $k_u > 0$ & $k_w > 0$ are predetermined gains
- r is the object location at every iteration I.e. $[x, y]^T$
- r_g is the goal position in $[x_g, y_g]^T$
- θ is the orientation of the vehicle w.r.t the goal position at every iteration.
- $\varphi \triangleq \arctan\left(F_y^\star / F_x^\star\right)$

This section is where all the predetermined variables are defined (such as the vehicle radius, the obstacle radius, the gains - k_u&k_w....), it also makes use of a matlab function called 'ginput' that allows you to specify start, end and obstacle location using a mouse left click. This allows one to visually inspect the placements of the points to ensure the minimum distance is kept between the obstacles. The script then goes on to call on the afore mentioned functions

in order to calculate linear velocity u and angular velocity ω of the vehicle. Substituting these values along with θ into eqn(1) gives $\dot{x}, \dot{y}, \dot{\theta}$, which are then used to update the position of the vehicle. The updated positions are displayed through a plot similar to the figures in the results section.

3.6 Results I

To demonstrate the algorithm, a similar environment was created using 10 static obstacles. Different start and end positions were tried while varying the obstacle positions at each time. These locations were obtained using matlab's ginput function, that allows you to click to create points on a 2-D Cartesian frame. The obstacle radius was the same as the object radius at 0.03 and k_u & k_w values at 0.015 & 2.5 respectively.

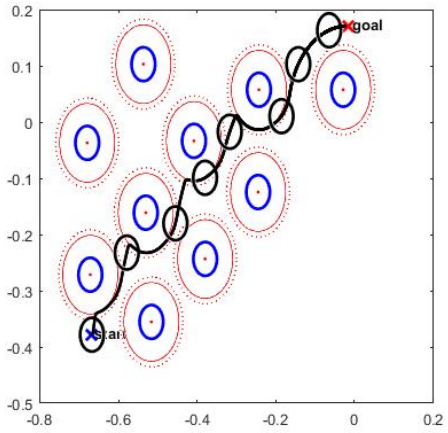

Figure 6: This is a caption

As you can see in fig (6), the vehicle is able to successfully navigate its way through all the obstacles in its path and approach the goal location.

- The blue 'x' mark represents the start position, while the red 'x' mark represents the goal position

- The black circle mirrors the region covered by the vehicle at specified iterations, and the black line is shows the position of the vehicle at every iteration.

- The red dot marks the position of all the obstacles, with the blue circle being the region covered by the obstacle

- The inner red circle defines the region Z, while the outer red circle defines the region F, with the region between them being the blending region of the attractive and repulsive vector fields

Note: Contact between the blue and black circle signifies that collision has occurred

3.7 Additional observation

If you observe the results of implementing the vector field algorithm, the vehicle follow a fixed vector field path that ultimately leads to the goal position as shown in the figures in the results section. This means that the agent sometimes takes a longer route than necessary. This led to the idea of rotating and translating the vector fields so that the path to the goal is always the shortest on the vector field. This was achieved by constantly changing the value of p in the attract_functn function at every iteration, which is similar to what was done in the deflect_function. A visualization of the idea can be seen in fig (7) & (8) below. Two similar scenarios were run using both ideas to illustrate the difference. On the left is a fixed attractive field, while the right shows the moving field idea.

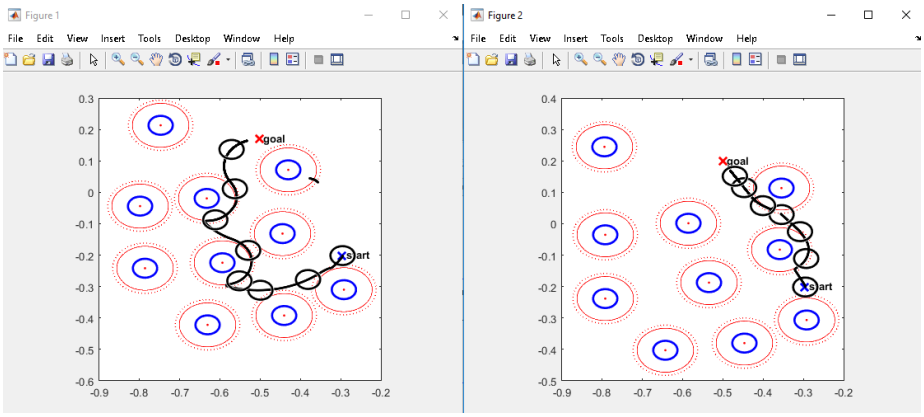

Figure 7: (a) fixed attractive field. (b) Moving attractive field

A seen in the figures above, the simulation supports the idea of moving the field, as figure (b) tends to get to the goal faster on both occasions.

Note: To further test and validate this idea, the moving field will be applied from now on

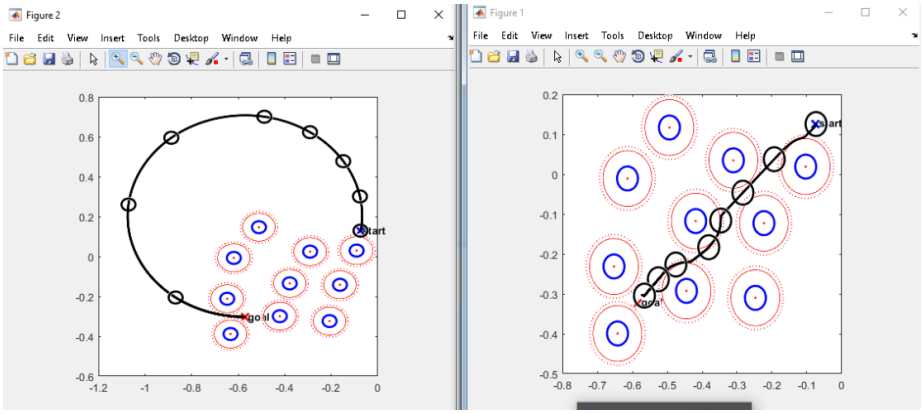

Figure 8: (a) fixed attractive field. (b) Moving attractive field

4 Implementation Part II

In this part the algorithm was scaled up to multiple vehicles. To test this, the code was slightly altered to simulate two vehicles in the same space. The revised code can be found in the same folder as this report. The structure is very similar to that of part I and is shown below, and all the initial variables remained the same. A more detailed explanation can be found in the matlab code

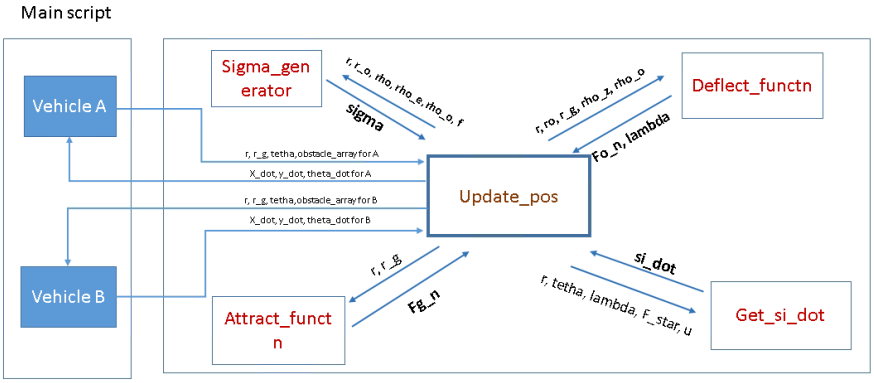

Figure 9: A flow diagram showing the matlab code layout for a multi-vehicle(2 in this case) and multiple obstacle case

4.1 Results II

Since the focus of this section was to test the scalability of the algorithm, it does not guarantee a collision free path between the vehicles. A similar scenario to that of section I was used in this section, with all the initial variables remaining the same.

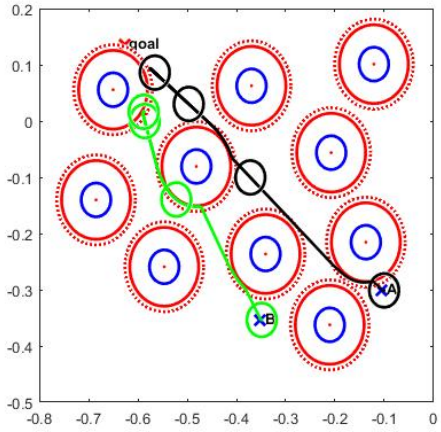

Figure 10: Two vehicles navigating through stationary obstacles to the same goal location.

Since the semi-cooperation algorithm is not implemented in this project, a simple instruction was set to stop one of the vehicles, while the other continued to avoid a collision. If you observe closely, you can see that the vehicle B(represented in green) actually slows down/ stops to keeps a safe distance from vehicle A(represented in black) to allow vehicle A to continue on its path. This, however, is only a minor fix, as it does not guarantee a collision-free space.

5 Implementation III

This section is the same as both sections I & II, but it takes into account moving obstacles. The change in location of the obstacle was simulated by a predefined velocity, to keep this simple all the obstacles were made to move at the same speed to avoid collision between them, and the new location of all the obstacles was updated to the vehicles. To avoid collision between the obstacles, and hence conflicting repulsive fields, a safe distance was maintained between the obstacles

by pre-defining a safe distance between all the obstacles and ensuring they all moved with the same velocity.

5.1 Results

The results from this contain both single vehicle and double vehicle simulation. Since the results are best visible through videos, the videos have been stored with their corresponding codes for convenience. On playing the videos, it can be observed that the vehicle can safely navigate through moving obstacles. This case is very sensitive to the velocity of the obstacles, as higher velocities do not give the vehicle enough time to react to the movement of the obstacles. In these cases, the obstacles were set to move at the velocity of $1.25*10^{-4}$units /iteration to ensure that the vehicles had enough time to avoid the obstacles.

6 Conclusion

The methodology presented in the earlier parts of the paper by (Panagou, 2014a) was successfully implemented, and this validates the concept of path planning for vehicles with unicycle kinematics in an environment with circular obstacles. This is made possible through the use of a family of vector fields that are attractive to a goal position and are repulsive around the obstacles. For this to work properly, a blending algorithm was devised based on the value of one parameter which was dependent on the distance of the current position of the vehicle to the obstacles. The paper also covers the expansion of this algorithm to multiple vehicles, the successful implementation of this idea is built on the notion of semi-cooperative path planning. i.e. managing conflicts between the vehicles to prevent collision and is beyond the scope of this project. Investigating the paper offers a solid insight into the concept of path planning, as it offers a good introduction to the concept of vector fields in path planning. It also opens ones mind to the vast applications of this concept, even beyond the aerospace industry. The progress up to this point has been well documented, with all the matlab codes stored in with the report and labeled accordingly for smooth transition, if this work is ever continued.

6.1 What's left

At this stage, only the cases that involve one vehicle have been successfully implemented, the other algorithms still to be implemented include:
1.) Multiple objects and multiple stationary obstacles
2.) Multiple objects and moving obstacles
3.) Investigate the effect of measurement noise on this concept.

7 Appendix A - Lessons learned

Here are a few things that were discovered while working with these algorithms:
- The algorithm is sensitive to angles and their orientation, as there were situations were the vehicle went of track as shown in fig(11). The error was later traced to the $(\psi - \theta)$ term in eqn(11), but was neutralized once this term and the θ term itself were capped between $-\pi \rightarrow \pi$.

- Another is the act that the code was not optimized for performance and runs on a few for loops that are dependent on the number of obstacles, and hence runs pretty slowly depending on how many obstacles are present.

- It is also important to change the path name for the moving obstacle cases, as it'll overwrite the previous file if it is not changes which could lead to loss of relevant data.

- There are four folders attached to this report, each folder name describes the function of the code inside, some folders have duplicate files, so some scripts are not in their folders. Follow the readme instructions provided in each folder as a guide in running each of the cases.

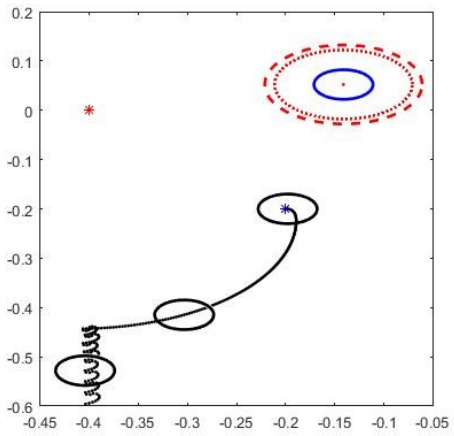

Figure 11: Error encountered

8 Appendix B - Some more scenarios

Here are more scenarios of the single vehicle/stationary obstacle case

Here are more scenarios of the multiple vehicle case/stationary obstacle

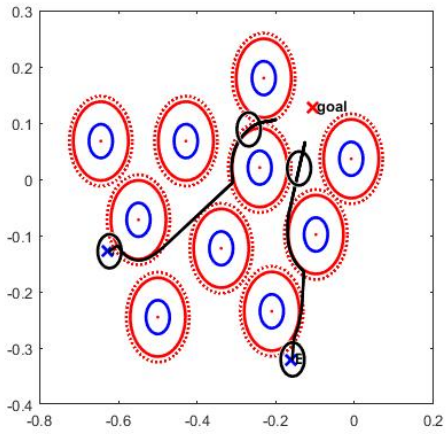

Figure 12: Multiple vehicle case 1

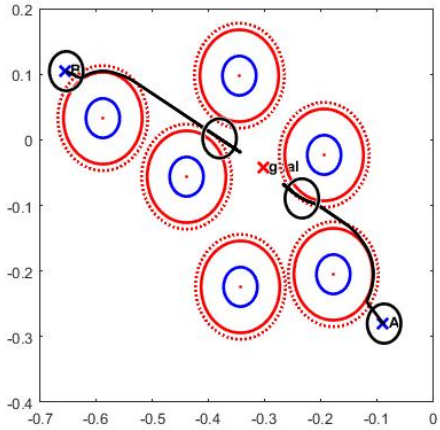

Figure 13: Multiple vehicle case 2

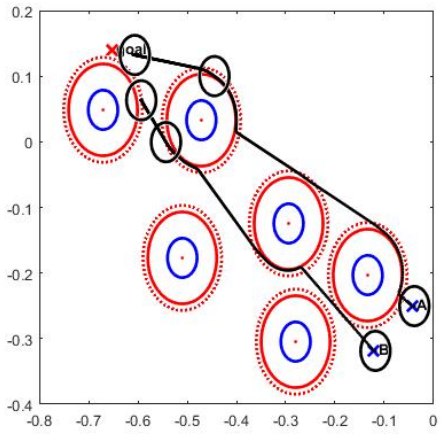

Figure 14: Multiple vehicle case 3

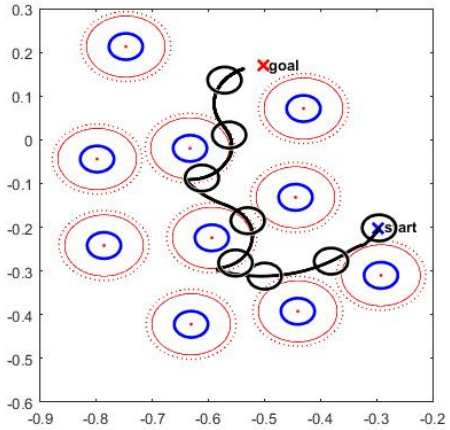

Figure 15: Single vehicle case 1

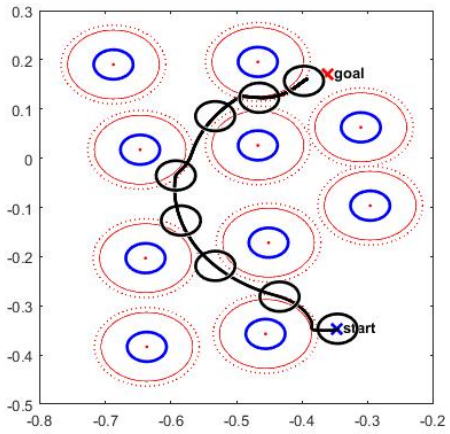

Figure 16: Single vehicle case 2

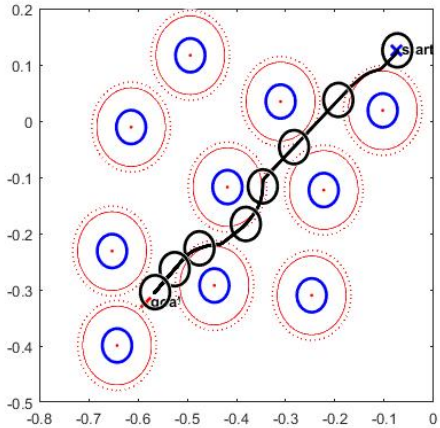

Figure 17: Single vehicle case 3

References

Garg, K., & Panagou, D. (2018, jan). A Robust Coordination Protocol for Safe Multi-Agent Motion Planning. In *2018 AIAA guidance navigation, and control conference.* American Institute of Aeronautics and Astronautics. Retrieved from `https://doi.org/10.2514%2F6.2018-0605` doi: 10.2514/6.2018-0605

Goldreich, P., & Kumar, P. (1990, nov). Wave generation by turbulent convection. *The Astrophysical Journal*, *363*, 694. Retrieved from `https://doi.org/10.1086%2F169376` doi: 10.1086/169376

Kumar, P., Goldreich, P., & Kerswell, R. (1994, may). Effect of nonlinear interactions on p-mode frequencies and line widths. *The Astrophysical Journal*, *427*, 483. Retrieved from `https://doi.org/10.1086%2F174159` doi: 10.1086/174159

Panagou, D. (2014a). Distributed Feedback Motion Planning Protocol for Multiple Unicycle Agents of Different Classes. *IEEE Transactions On Automatic Control*, *62(3)*.

Panagou, D. (2014b). A Distributed Feedback Motion Planning Protocol for Multiple Unicycle Agents of Different Classes. *IEEE Transactions On Automatic Control*, *62(3)*.