Using Variational Calculus To Model Paths Around Obstacles

Lance Remigio, Jin Zhang, Fatma Djellouli, Woody Lee

September 16, 2023

Contents

Chapter 1

Problem Statement

There are N number of robots located at position $x_i \in \mathbb{R}^2, i = 1, ..., N$. They need to be transferred to some given target position $y_i \in \mathbb{R}^2, i = 1, ..., N$. There are M circular obstacles in the domain of radius $R > 0$ and centers $c_i, i = 1, ..., M$, which the robots must avoid. Additionally, the robots should not collide with each other. Find optimal paths for the robots, so that their velocities over the trajectory is minimized with these objectives in mind.

Abstract

In order to model this situation as an optimization problem, we develop a vector valued cost functional. The minimization of this cost functional should produce optimal trajectories for the robots according to the conditions of the problem. In order to motivate this cost function, we divide it's derivation into three sections, one for each of the problem's conditions. The final functional is a sum of these three sub-parts. Then, we go on to calculate the first order necessary conditions for optimality, followed by numerically solving the resulting differential equations. Lastly, we consider a few different scenarios to verify that our model provides a reasonable representation of the trajectories.

Notation

- $P_j(t)$:
  - Position of robot j at time t.
  - $P_j(t) = (x_j(t), y_j(t))$ for $j = 1, .., N$.
  - Assume $t \in [0, 1]; t_f = 1$
- P_j^0 :

  - Initial position of robot j.
  - $P_j^0 = (x_j^0, y_j^0)$ for $j = 1, .., N$.
  - Note that: $(x_j^0, y_j^0) = (x_j(0), y_j(0))$.

- P_j^F :
  - Final position of robot j.
  - $P_j^F = (x_j^F, y_j^F)$ for $j = 1, .., N$.
  - Note that $P_j^F = (x_j^F, y_j^F) = (x_j(1), y_j(1))$.

- C_ℓ :
  - Center of Obstacle ℓ where $C_\ell = (a_\ell, b_\ell)$ for $\ell = 1, ..., M$

- $P(t) = (P_1(t), P_2(t), ..., P_N(t))$

- $C = (C_1, C_2, ..., C_M)$

Chapter 2

Theory

2.1 Modelling the Trajectory travelled by each robot

We know that the length of the trajectory travelled by the robot can be represented by the following integral:

$$\begin{aligned} L_j &= \int_0^1 |P_j'(t)| \; dt \text{ for } j = 1, ..., N \\ &= \int_0^1 \sqrt{\left(\frac{dx_j}{dt}\right)^2 + \left(\frac{dy_j}{dt}\right)^2}. \end{aligned}$$

We can use the equation above to find the total length of the trajectories travelled by N robots. Hence, we have that

$$\begin{aligned} L(P_1, .., P_N) &= \sum_{j=1}^{N} L_j = \sum_{j=1}^{N} \left[\int_0^1 |P_j'(t)| \; dt \right] \\ \Rightarrow L(P_1, .., P_N) &= \int_0^1 \left[\sum_{j=1}^{N} |P_j'(t)| \right] dt. \end{aligned}$$

Hence,

$$\begin{aligned} L(P_1, .., P_N) &= \int_0^1 |P'(t)| \; dt. \\ \text{where } |P'(t)| &= \sum_{j=1}^{N} \left| \frac{dP_j}{dt} \right| \\ &= \sum_{j=1}^{N} \sqrt{\left(\frac{dx_j}{dt}\right)^2 + \left(\frac{dy_j}{dt}\right)^2}. \end{aligned}$$

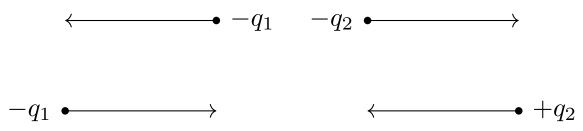
Minimizing this functional will create the shortest trajectory for each robot. Given that the robots must travel the distance within the time period [0,1], it follows that the shortest trajectories will also result in the smallest required velocities. This is because the longer the path, the faster the robot must move to get from start point to end point in the set time.

2.2 Modelling Repulsion Between Robots

In this section, we will use Coulomb's law to model our situation.

2.2.1 Background on Coulomb's Law

- Let q_1 and q_2 be two charges. They experience an electrostatic force of attraction or repulsion.



- The magnitude of the electrostatic force is inversely proportional to the square of the distance between them:

$$|F| = K\frac{|q_1||q_2|}{r^2}$$

  where $|F|$ represents the magnitude of force, K is Coulomb's constant, q_1, q_2 are the particle charges, and r is the distance between q_1 and q_2.

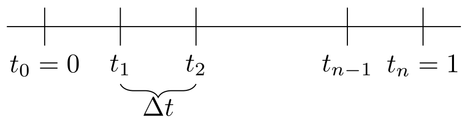
Note that the closer the two particles are, the larger the force. We incorporate this idea into our situation by viewing each robot as a moving particle with charge q_j. We want to minimize the force between them which results in maximizing the distance between them, therefore preventing collision.

Our assumption is that all the robots are identical. Therefore, they have the same charge q. Hence, at a fixed time t, the magnitude of the force F_j between each robot i and j is:

$$|F_{ij}| = K\frac{Q^2}{|P_i(t) - P_j(t)|^2}.$$

But, we need to evaluate the force F_{ij} over continuous time.

We consider the following equal subdivision of the travel time interval $[0, 1]$:



where we have n subintervals $[t_\ell, t_{\ell+1}]$ and $\Delta t = \frac{1-0}{n} = \frac{1}{n}$. We assume that every time step Δt is small enough that the force is constant for it's duration. So, we have,

$$F_{ij}(t) = K\frac{Q^2}{|P_i(t_\ell) - P_j(t_\ell)|^2} \text{ for all } t \in [t_\ell, t_{\ell+1})].$$

Hence, the total force over all the intervals is:

$$\begin{aligned} F_{ij} &= \sum_{\ell=0}^{n} F_{ij}(t_\ell)\Delta t \\ &= \frac{1}{n}\sum_{\ell=0}^{N} F_{ij}(t_\ell). \end{aligned}$$

Observe that as that as $N \to +\infty$, this Riemann sum converges to

$$\frac{1}{n}\sum_{\ell=0}^{N} F_{ij}(t_\ell) \to \int_0^1 F_{ij}(t)\ dt.$$

Hence, the magnitude of the total force between robot i and j during their trip is given by:

$$F_{ij} = KQ^2\int_0^1 \frac{dt}{|P_i(t) - P_j(t)|^2}.$$

To make calculations much simpler, we set the $KQ^2 = 1$ since they are just constants. Hence, the total force between each robot is given by:

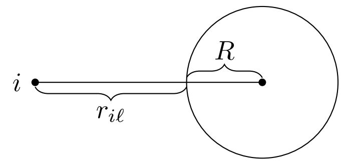
$$F = \frac{1}{2}\left(\sum_{i,j=1}^{N} F_{ij}\right)$$

where $i \neq j$. Hence,

$$F(P) = F(P_1, ..., P_N) = \frac{1}{2}\int_0^1 \sum_{i,j=1}^{N} F_{ij}\ dt.$$

2.3 Modelling Repulsion Between Robots and Obstacles

In this section, we will create a functional that models repulsion of robots from obstacles. Our goal is to minimize the likelihood that any robot will hit obstacles. Following the principles presented in the previous section, we assume that all the obstacles have the same charge



where $r_{i\ell}$ represents the distance between robot and outside of obstacle and R is the radius of obstacle. Hence,

$$|r_{i\ell}(t)| = \sqrt{(x_i(t) - a_\ell)^2 + (y_i(t) - b_\ell)^2} - R$$

and using a similar procedure to the previous section, we arrive at:

$$G = \int_0^1 \left(\sum_{i=1}^{N} \sum_{\ell=1}^{M} \frac{1}{|r_{i\ell}(t)|^2} \right) dt.$$

So, in order to avoid hitting obstacles, we want to minimize the functional

$$G(P) = G(P_1, ..., P_N) = \int_0^1 \left(\sum_{i=1}^{N} \sum_{\ell=1}^{M} \frac{1}{|r_{i\ell}(t)|^2} \right) dt.$$

Chapter 3

Derivations

Our goal is to minimize the functional $\tilde{L}(P)$ such that:

$$\min_{P \in D} \tilde{L}(P)$$

where:

$$\tilde{L}(P) = L(P) + F(P) + G(P)$$

in which $L(P)$ represents the total length of trajectories, $F(P)$ represents the total force between each robot, and $G(P)$ represents the total force between each robot and each obstacle. To determine the appropriate space D, we evaluate the Gateaux Derivative of $\tilde{L}(P)$.

The Gateaux derivative of $\delta\tilde{L}(P;V)$ is

$$\delta\tilde{L}(P;V) = \delta L(P;V) + \delta F(P;V) + \delta G(P;V)$$

where $V = (v_1, ..., v_n)$ and $v_j = (u_j, w_j)$.

We will compute the Gateaux derivatives $\delta L(P;V), \delta F(P;V), \delta G(P;V)$ separately.

3.0.1 The Gateaux Derivative of $L(P)$

Proposition 1. The Gateaux Derivative of $L(P)$ is

$$\delta L(P;v) = \int_0^1 \tilde{P}' \cdot V' \, dt.$$

where

$$\tilde{P}' = \left(\frac{P_1'}{|P_1'|}, \frac{P_2'}{|P_2'|}, ..., \frac{P_N'}{|P_N'|} \right).$$

Proof. Expand $L(P)$ to get

$$L(P) = \int_0^1 |P'(t)| \; dt = \int_0^1 \sum_{j=1}^{N} |P_j'(t)| \; dt.$$

Define $g(\varepsilon) = L(P + \varepsilon V)$. We can write $g(\varepsilon)$ as

$$\begin{aligned} g(\varepsilon) &= \int_0^1 \sum_{j=1}^{N} |P_j'(t) + \varepsilon V_j'| \; dt \\ &= \int_0^1 \sum_{j=1}^{N} \sqrt{(x_j' + \varepsilon u_j')^2 + (y_2' + \varepsilon w_j')^2} \; dt. \end{aligned}$$

Computing $g'(\varepsilon)$, then gives us

$$\begin{aligned} g'(\varepsilon) &= \frac{d}{d\varepsilon} \int_0^1 \sum_{j=1}^{N} \sqrt{(x_j' + \varepsilon u_j')^2 + (y_j' + \varepsilon w_j')^2} \; dt \\ &= \int_0^1 \sum_{j=1}^{N} \frac{\partial}{\partial \varepsilon} \sqrt{(x_j' + \varepsilon u_j')^2 + (y_j' + \varepsilon w_j')^2} \; dt \\ &= \int_0^1 \sum_{j=1}^{N} \frac{(x_j' + \varepsilon u_j')u_j' + (y_j' + \varepsilon w_j')w_j'}{\sqrt{(x_j' + \varepsilon u_j')^2 + (y_j' + \varepsilon w_j')^2}} \; dt. \end{aligned}$$

Taking the limit as $\varepsilon \to 0$, then gives us

$$\begin{aligned} g'(0) &= \int_0^1 \sum_{j=1}^{N} \frac{x_j' u_j' + y_j' w_j'}{\sqrt{(x_j')^2 + (y_j')^2}} \; dt \\ &= \int_0^1 \sum_{j=1}^{N} \frac{P_j' V_j'}{|P_j'|} \; dt \end{aligned}$$

where $x_j' u_j' + y_j' w_j' - P_j' \cdot V_j'$ and $|P_j'| = \sqrt{(x_j')^2 + (y_j')^2}$. Hence, applying the Gateaux Derivative to the total length of all possible trajectories is

$$\delta L(P; v) = \int_0^1 \tilde{P}' \cdot V' \; dt.$$

where

$$\tilde{P}' = \left(\frac{P_1'}{|P_1'|}, \frac{P_2'}{|P_2'|}, ..., \frac{P_N'}{|P_N'|} \right).$$

■

3.0.2 The Gateaux Derivative of F(P)

Proposition 2. The Gateaux Derivative of the total force between each robot is

$$\delta F(P;V) = \int_0^1 \sum_{i=2}^{N}\sum_{j=1}^{i-1} \frac{2(P_i - P_j)\cdot(v_i - v_j)}{|P_i - P_j|^4}\, dt.$$

Proof. First we expand $F(p)$ to get

$$\begin{aligned} F(P) &= \frac{1}{2}\int_0^1 \sum_{i,j=1}^{N} F_{ij}\ dt \\ &= \int_0^1 \sum_{i=2}^{N}\sum_{j=1}^{i-1} F_{ij}\ dt \\ &= \int_0^1 \sum_{i=2}^{N}\sum_{j=1}^{i-1} \frac{dt}{|P_i(t) - P_j(t)|^2}. \end{aligned}$$

Define $g(\varepsilon) = F(P + \varepsilon V)$, we compute

$$\begin{aligned} g(\varepsilon) &= \int_0^1 \sum_{i=2}^{N}\sum_{j=1}^{i-1} \frac{dt}{|(P_i + \varepsilon v_i) - (P_j + \varepsilon v_j)|^2} \\ &= \int_0^1 \sum_{i=2}^{N}\sum_{j=1}^{i-1} \frac{dt}{[(x_i + \varepsilon v_i) - (x_j + \varepsilon u_j)]^2 + [(y_i + \varepsilon w_i) - (y_j + \varepsilon w_j)]^2} \\ &= \int_0^1 \sum_{i=2}^{N}\sum_{j=1}^{i-1} \frac{dt}{[(x_i - x_j) + \varepsilon(u_i - u_j)]^2 + [(y_i - y_j) + \varepsilon(w_i - w_j)]^2}. \end{aligned}$$

We can take the derivative with respect to ε in the integrand will give us the following expression

$$\begin{aligned} g'(\varepsilon) &= \frac{d}{d\varepsilon}\int_0^1 \sum_{i=2}^{N}\sum_{j=1}^{i-1} \frac{dt}{[(x_i - x_j) + \varepsilon(u_i - u_j)]^2 + [(y_i - y_j) + \varepsilon(w_i - w_j)]^2} \\ &= \int_0^1 \sum_{i=2}^{N}\sum_{j=1}^{i-1} \frac{\partial}{\partial\varepsilon}\left[\frac{1}{[(x_i - x_j) + \varepsilon(u_i - u_j)]^2 + [(y_i - y_j) + \varepsilon(w_i - w_j)]^2}\right] dt \\ &= \int_0^1 \sum_{i=2}^{N}\sum_{j=1}^{i-1} \frac{2[(x_i - x_j) + \varepsilon(u_i - u_j)](u_i - u_j) + 2[(y_i - y_j) + \varepsilon(w_i - w_j)](w_i - w_j)}{\Big[[(x_i - x_j) + \varepsilon(u_i - u_j)]^2 + [(y_i - y_j) + \varepsilon(w_i - w_j)]^2\Big]^2}\ dt. \end{aligned}$$

Letting $\varepsilon \to 0$ and recalling that $v_i = (u_i, w_i)$ and $v_j = (u_j, w_j)$, we have that

$$\begin{aligned}\lim_{\varepsilon\to 0} g'(\varepsilon) &= \int_0^1 \sum_{i=2}^{N}\sum_{j=1}^{i-1} \frac{2(x_i - x_j)(u_i - u_j) + 2(y_i - y_j)(w_i - w_j)}{[(x_i - x_j)^2 + (y_i - y_j)^2]^2}\, dt \\ &= \int_0^1 \sum_{i=2}^{N}\sum_{j=1}^{i-1} \frac{2(P_i - P_j)\cdot(v_i - v_j)}{|P_i - P_j|^4}\, dt.\end{aligned}$$

■

Hence, the Gateaux Derivative of the total force $F(p)$ is

$$\delta F(P; V) = \int_0^1 \sum_{i=2}^{N}\sum_{j=1}^{i-1} \frac{2(P_i - P_j)\cdot(v_i - v_j)}{|P_i - P_j|^4}\, dt.$$

3.0.3 Gateaux Derivative of $G(P)$

Proposition 3. Show that the Gateaux Derivative of $G(P)$ is

$$\delta G(P; v) = \int_0^1 \sum_{i=1}^{N}\sum_{\ell=1}^{M} \frac{-2(P_i - C_\ell \cdot V_i)}{|r_{i\ell}|^3 |P_i(t) - C_\ell|}\, dt.$$

Proof. Expanding $G(P)$, we get

$$\begin{aligned}G(P) &= \int_0^1 \sum_{i=1}^{N}\sum_{\ell=1}^{M} \frac{dt}{|r_{i\ell}(t)|^2} \\ &= \int_0^1 \sum_{i=1}^{N}\sum_{\ell=1}^{M} \frac{dt}{(\sqrt{(x_i(t) - a_\ell)^2 + (y_i(t) - b_\ell)^2} - R)^2}.\end{aligned}$$

Define $g(\varepsilon) = G(P + \varepsilon V)$ and write

$$g(\varepsilon) = \int_0^1 \sum_{i=1}^{N}\sum_{\ell=1}^{M} \frac{dt}{(\sqrt{((x_i + \varepsilon u_i) - a_\ell)^2 + ((y_i + \varepsilon w_i) - b_\ell)^2} - R)^2}.$$

with the Gateaux derivative being

$$g'(\varepsilon) = \int_0^1 \sum_{i=1}^{N}\sum_{\ell=1}^{M} \left[-2 \frac{2((x_i + \varepsilon u_i) - a_\ell)u_i + 2((y_i + \varepsilon w_i) - b_\ell)w_i}{|r_{i\ell}(t)|^3 2\sqrt{((x_i + \varepsilon u_i) - a_\ell)^2 + ((y_i + \varepsilon w_i) - b_\ell)^2}} \right] dt.$$

Letting $\varepsilon \to 0$, we can write

$$\begin{aligned}\lim_{\varepsilon\to 0} g'(\varepsilon) &= \int_0^1 \sum_{i=1}^{N}\sum_{\ell=1}^{M} \frac{-2[(x_i - a_\ell)u_i + (y_i - b_\ell)w_i]}{|r_{i\ell}|^3 |P_i(t) - C_\ell|}\, dt \\ &= \int_0^1 \sum_{i=1}^{N}\sum_{\ell=1}^{M} \frac{-2(P_i - C_\ell)\cdot V_i}{|r_{i\ell}|^3 |P_i(t) - C_\ell|}\, dt.\end{aligned}$$

Hence, we have

$$\delta G(P;V) = \int_0^1 \sum_{i=1}^{N}\sum_{\ell=1}^{M} \frac{-2(P_i - C_\ell)\cdot V_i}{|r_{i\ell}|^3 |P_i(t) - C_\ell|}\, dt.$$

■

3.0.4 Result

Putting everything together, we end up with

$$\begin{aligned}\delta L(\tilde{P};V) &= \delta L(P;V) + \delta F(P;V) + \delta G(P;V) \\ &= \int_0^1 \left[\sum_{j=1}^{N} \frac{P_j' \cdot V_j'}{|P_j'|} + \sum_{i=2}^{N}\sum_{j=1}^{i-1} \frac{2(P_i - P_j)\cdot(V_i - V_j)}{|P_i - P_j|^4} - \sum_{i=1}^{N}\sum_{\ell=1}^{M} \frac{2(P_i - C_\ell)\cdot V_i}{\left|r_{i\ell}\right|^3 \left|P_i(t) - C_l\right|} \right] dt.\end{aligned}$$

Chapter 4

First Order Necessary Conditions for Optimality

4.1 Optimality

This functional is defined over the set

$$D = \{P = (P_1, ..., P_N) \in (C^1[0,1])^{2N}; P(0) = P^o, P(1) = P^F\}$$

where

$$P^0 = (P_1^0, ..., P_N^0) P_j^0 = (x_j^0, y_j^0) \text{ with } j = 1, ..., N$$
$$P^F = (P_1^F, ..., P_N^F) P_j^F = (x_j^F, y_j^F) \text{ with } j = 1, ..., N$$

and the admissible directions are

$$D_0 = \{V = (V_1, ..., V_N) \in (C^1[0,1])^{2N}; v(0) = v(1) = 0\}$$

To conclude, we are going to use Proposition 4.2 (page 98 of textbook) which states

Proposition 4. If we have functions $g, h \in C[0,1]$ and

$$\int_0^1 g(x)v(x) + h(x)v'(x) = 0$$

for all v in

$$D_0 = \{V = (V_1, ..., V_N) \in (C^1[0,1])^{2N}; v(0) = v(1) = 0\}$$

then, $h' = g$.

Applying the above proposition to a vector function involves looking at the component level:

Take $V = (V_1, ..., V_N) \in D_0$ with $V_2 = V_3 = ... = V_N = 0$. Note that $V_1 \in (C[0,2])^2$ where the boundary conditions are $V_1(0) = V_1(1) = 0$. We substitute such V into $\delta\tilde{L}(P;V)$

$$\delta\tilde{L}(P;V) = \int_0^1 \left[\frac{P_1' \cdot V_1'}{|P_1'|} - \sum_{i=2}^{N} \frac{2(P_i - P_1)\cdot v_1}{|P_i - P_1|^4} - \sum_{\ell=1}^{M} \frac{2(P_1 - C_\ell)\cdot V_1}{|r_{i\ell}|^3 |P_1 - C_\ell|}\right] dt$$

Choose $v_1 = (u_1(t), 0)$ with $u_1(0) = u_1(1) = 0$. Substituting,

$$\begin{aligned}\delta\tilde{L}(P;V) &= \int_0^1 \frac{x_1'}{|P_1'|}u_1' - \sum_{i=2}^{N} \frac{2(x_i - x_1)u_1}{|P_i - P_1|^4} - \sum_{\ell=1}^{M} \frac{2(x_1 - a_\ell)u_1}{|r_{i\ell}|^3 |P_1 - C_\ell|}\ dt \\ &= \int_0^1 \frac{x_1'}{|P_1'|}u_1' + \left(\sum_{i=2}^{N} \frac{-2(x_i - x_1)}{|P_i - P_1|^4} + \sum_{\ell=1}^{M} \frac{-2(x_1 - a_\ell)}{|r_{i\ell}|^3 |P_1 - C_\ell|}\right)u_1\ dt = 0.\end{aligned}$$

Hence, we can now apply our proposition since our integral is now in the form $\int hu_1' + gu_1'\ dt = 0$. Note that we can also apply the same computation but this time taking $v_1 = (0, w_1(t))$ with $w_1(0) = w_1(1) = 0$. The only difference with this procedure is replacing x_i by y_i and a_ℓ by b_ℓ. Hence, we have for robot 1, a coupled differential equation:

$$\begin{aligned}\frac{d}{dt}\left(\frac{x_1'}{P_1'}\right) &= \sum_{i=2}^{N} \frac{-2(x_i - x_1)}{|P_i - P_1|^4} + \sum_{\ell=1}^{M} \frac{-2(x_1 - a_\ell)}{|r_{i\ell}|^3 |P_1 - C_\ell|}, \\ \frac{d}{dt}\left(\frac{y_1'}{P_1'}\right) &= \sum_{i=2}^{N} \frac{-2(y_i - y_1)}{|P_i - P_1|^4} + \sum_{\ell=1}^{M} \frac{-2(y_1 - b_\ell)}{|r_{i\ell}|^3 |P_1 - C_\ell|}.\end{aligned}$$

Lastly, we can follow this same procedure with $V = (0, V_2, 0, ...0))$ and then $V = (0, 0, V_3, 0, ..., 0)$ up to $V = (0, 0, ..., 0, V_N)$. For each V, we derive two differential equations of the same form. The generalization of this process is as follows:

The Gateaux derivative for each jth choice of V where $V = (0, 0, ...V_j, ..., 0, 0)$ is

$$\delta\tilde{L}(P;V) = \int_0^1 \left[\frac{P_j' \cdot V_j'}{|P_j'|} + \sum_{i=1}^{N} \frac{2(P_i - P_j)\cdot(-v_j)}{|P_i - P_j|^4} + \sum_{\ell=1}^{M} \frac{-2(P_j - C_\ell)\cdot V_i}{|r_{i\ell}|^3 |P_j(t) - C_l|}\right] dt$$

Plugging two cases of $v_j = (u_j, 0)$ and $V_j = (0, w_j)$ followed by Proposition 4.2, we get the general system:

$$\frac{d}{dt}\left(\frac{x'_j}{P'_j}\right) = \sum_{i=1}^{N} \frac{-2(x_i - x_j)}{|P_i - P_j|^4} + \sum_{\ell=1}^{M} \frac{-2(x_j - a_\ell)}{|r_{i\ell}|^3 |P_j - C_\ell|},$$
$$\frac{d}{dt}\left(\frac{y'_j}{P'_j}\right) = \sum_{i=1}^{N} \frac{-2(y_i - y_j)}{|P_i - P_j|^4} + \sum_{\ell=1}^{M} \frac{-2(y_j - b_\ell)}{|r_{i\ell}|^3 |P_j - C_\ell|}$$

for $j = 1, .., N$ and $j = 1, .., N$ and $i \neq j$. Compacting this into a single equation, we get the following 1st order necessary condition:

$$\frac{d}{dt}\left(\frac{P'_j(t)}{|P_j(t)|}\right) = \sum_{i=1}^{N} \frac{-2(P_i(t) - P_j(t))}{|P_i(t) - P_j(t)|^4} + \sum_{l=1}^{M} \frac{-2(P_j(t) - C_\ell)}{|r_{i\ell}(t)|^3 |P_j(t) - C_\ell|}$$

for where $j = 1, .., N$ and $i \neq j$.

Chapter 5

Test Cases and Summary

5.1 Sample Cases

For the purposes of the computation, we simplify our cost functional and use the squared length instead of true length in L in order to accommodate the coding format. Even with this simplification, the model is robust and provides a reasonable representation of the trajectories. This is evidenced by the following four test cases:

1. Two robots with parallel destination.
2. Two robots with paths that intersect.
3. Two robots and one obstacle.
4. Two or more robots with two or more obstacles.

Below is our algorithm we used to test these cases (for the 2 robots and 1 obstacle case; more robots and obstacles cases are similar):

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as int
# weights
# Repulsiveness between of robots
C1 = 1
# Repulsiveness of obstacles
C2 = 3
# Robots
robot1_start = #insert initial position for robot 1
robot1_end = #insert final position for robot 1
```

```
robot2_start = #insert initial position for robot 2

robot2_end = #insert final position for robot 2

#

# Obstacles

obs1_radius = #specify size of radius of obstacle

obs1_pos = #specify location of center of obstacle

#

robot1_x = #insert guess for x position for robot 1

robot1_y = #insert guess for y position for robot 1

robot2_x = #insert guess for x position for robot 2

robot2_y = #insert guess for y position for robot 2

#Model

def dxydt(t, x_y):

x1, y1, x2, y2, dx1, dy1, dx2, dy2 = x_y


x1_el = -(C1 * (x1 - x2)) / ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 2 - C2 * (x1 - 2)


/ (np.sqrt((x1 - obs1_pos[0]) ** 2 + (y1 - obs1_pos[1]) ** 2) - obs1_radius) ** 3


/ np.sqrt((x1 - obs1_pos[0]) ** 2 + (y1 - obs1_pos[1]) ** 2)


y1_el = -(C1 * (y1 - y2)) / ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 2 - C2 * (y1 - 2)


/ (np.sqrt((x1 - obs1_pos[0]) ** 2 + (y1 - obs1_pos[1]) ** 2) - obs1_radius) ** 3


/ np.sqrt((x1 - obs1_pos[0]) ** 2 + (y1 - obs1_pos[1]) ** 2)


x2_el = (C1 * (x1 - x2)) / ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 2 - C2 * (x2 - 2)


/ (np.sqrt((x2 - obs1_pos[0]) ** 2 + (y2 - obs1_pos[1]) ** 2) - obs1_radius) ** 3


how / np.sqrt((x2 - obs1_pos[0]) ** 2 + (y2 - obs1_pos[1]) ** 2)


y2_el = (C1 * (y1 - y2)) / ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 2 - C2 * (y2 - 2)


/ (np.sqrt((x2 - obs1_pos[0]) ** 2 + (y2 - obs1_pos[1]) ** 2) - obs1_radius) ** 3


/ np.sqrt((x2 - obs1_pos[0]) ** 2 + (y2 - obs1_pos[1]) ** 2)
```

```
return dx1, dy1, dx2, dy2, x1_el, y1_el, x2_el, y2_el



# Boundary Conditions
def bc(ya, yb):
return ya[0] - robot1_start[0], yb[0] - robot1_end[0] , ya[1] - robot1_start[1]
, yb[1] - robot1_end[1], ya[2] - robot2_start[0]
, yb[2] - robot2_end[0], ya[3] - robot2_start[1], yb[3] - robot2_end[1]

t_guess = np.linspace(0, 1, 5)
xy_guess = np.zeros((8, t_guess.size))
xy_guess[0] = robot1_x
xy_guess[1] = robot1_y
xy_guess[2] = robot2_x
xy_guess[3] = robot2_y
res = int.solve_bvp(dxydt, bc, t_guess, xy_guess)
```

In addition, we have the following code designed to test the cases we outlined in the beginning of this section.

```
#Plots for the paths of robots 1 and 2:

t = np.linspace(0, 1, 201)
obs_t = np.linspace(0, 2 * np.pi, 51)
plt.plot(res.sol(t)[0], res.sol(t)[1], label = 'Robot 1')
plt.plot(res.sol(t)[2], res.sol(t)[3], label = 'Robot 2')
plt.plot(obs1_radius * np.cos(obs_t) + obs1_pos[0], obs1_radius * np.sin(obs_t)
    obs1_pos[1], label = 'obstacle')
plt.legend()

#Plots that verify collision free paths between robots:
t = np.linspace(0, 1, 101)
plt.plot(t, res.sol(t)[0], label = 'X-Robot 1')
plt.plot(t, res.sol(t)[1], label = 'Y-Robot 2')
plt.plot(t, res.sol(t)[2], label = 'X-Robot 1')
plt.plot(t, res.sol(t)[3], label = 'Y-Robot 2')
plt.legend()
```

5.1.1 Two Robots With Parallel Destination

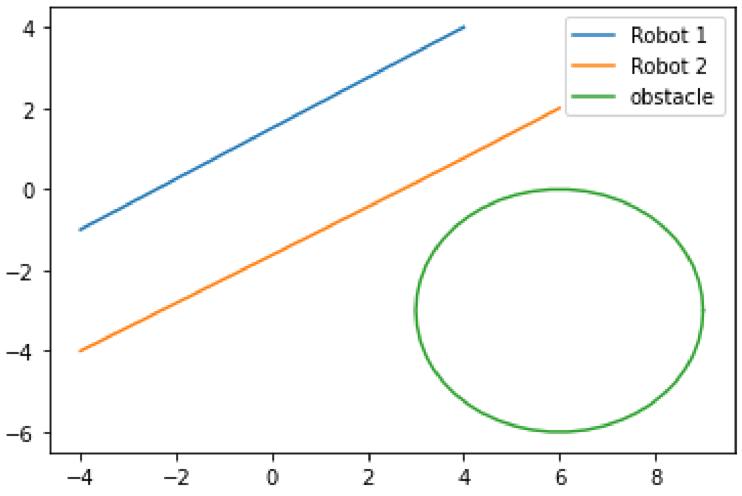


Figure 5.1: Robot 1 travelling from (-4,-1) to (4,4) and robot 2 from (-4,-4) to (6,2) with obstacle centered at (6,-3) with a radius $R = 3$.

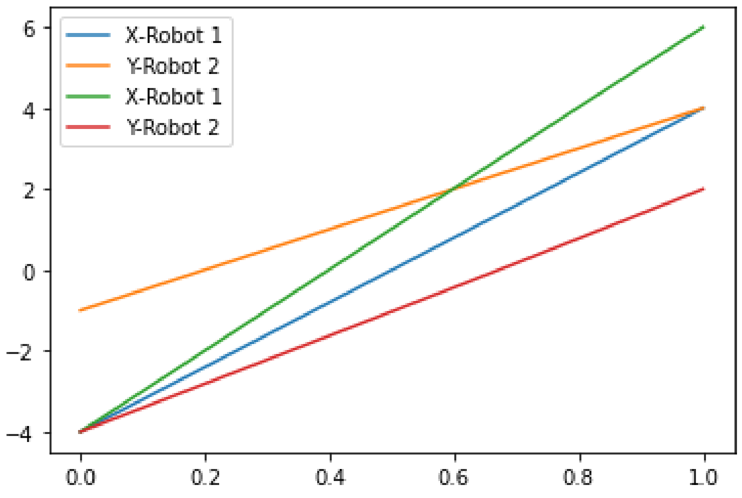


Figure 5.2: Collision free path from figure 1

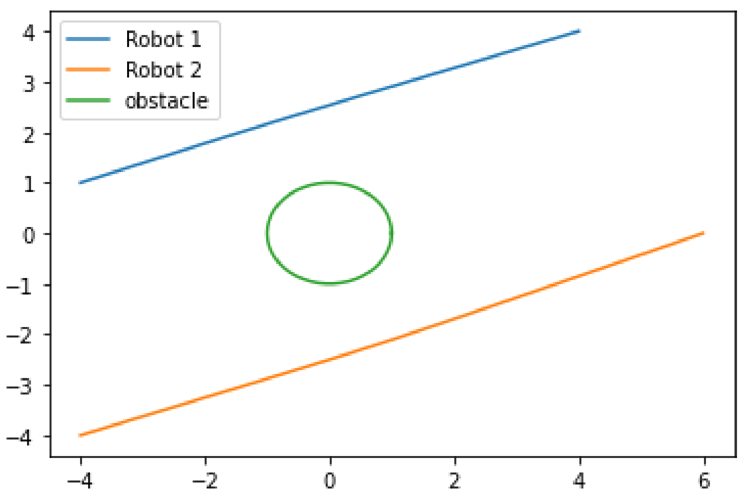


Figure 5.3: Robot 1 with position (-4,1) at $t = 0$ and position (4,4) at $t = 1$ and robot 2 with a position at (-4,-4) at $t = 0$ and position (6,0) at $t = 1$. Obstacle centered at (0,0) with radius $R = 1$.

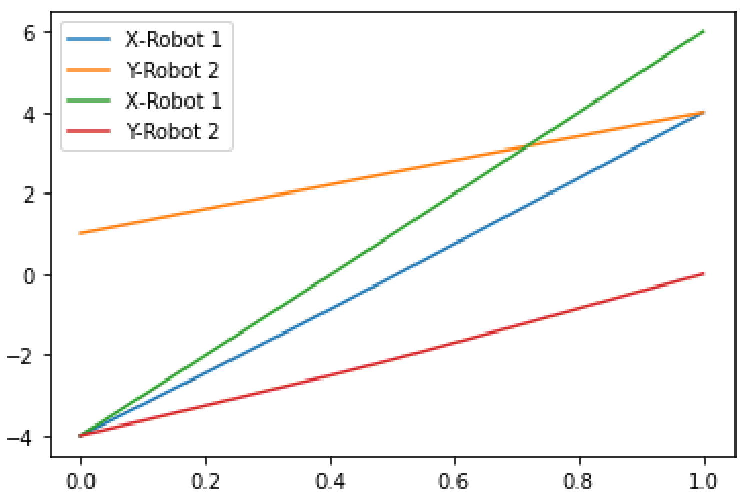


Figure 5.4: Collision free path from figure 3

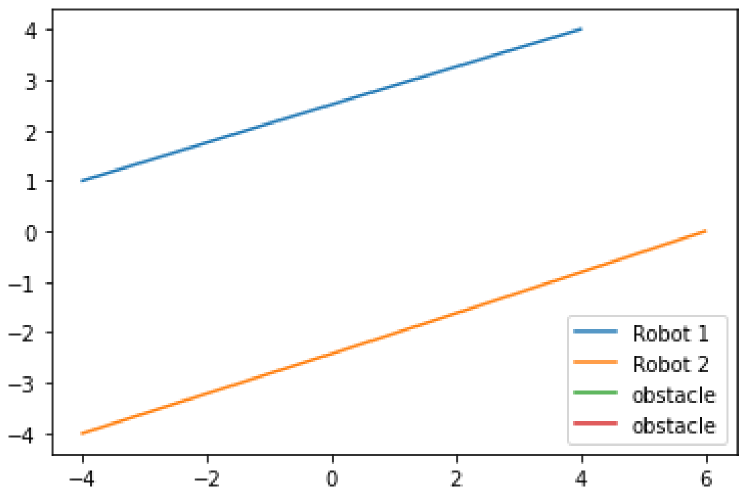


Figure 5.5: Robot 1 with position (-4,1) at $t = 0$ and position (4,4) at $t = 1$ and robot 2 with a position at (-4,4) at $t = 0$ and position (6,0) at $t = 1$ with **no obstacle**.

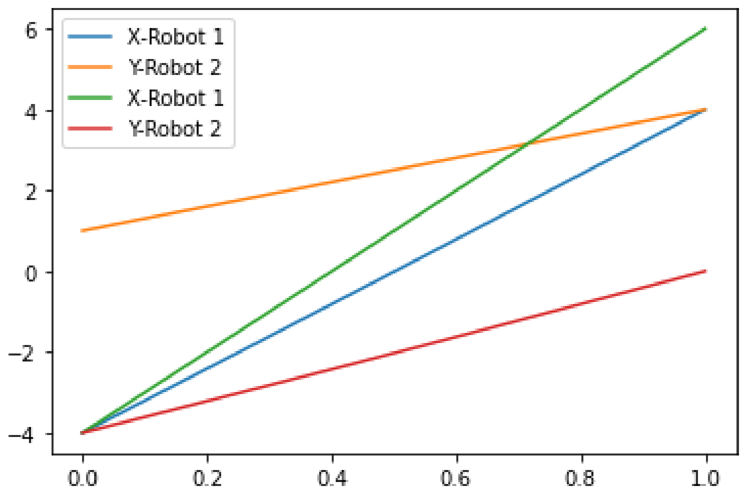


Figure 5.6: Collision free path from figure 5

5.1.2 Two Robots With Paths That Intersect

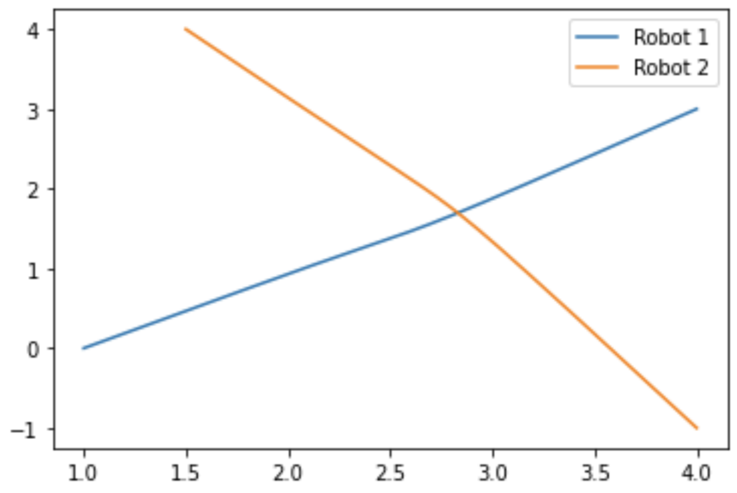


Figure 5.7: Robot 1 with position (1,0) at t = 0 and position (4,3) at t = 1 and robot 2 with position at (4,-1) at t = 0 and position (1.5,4) at t = 1 with **no obstacle**.

5.1.3 Two Robots With Paths That Intersect

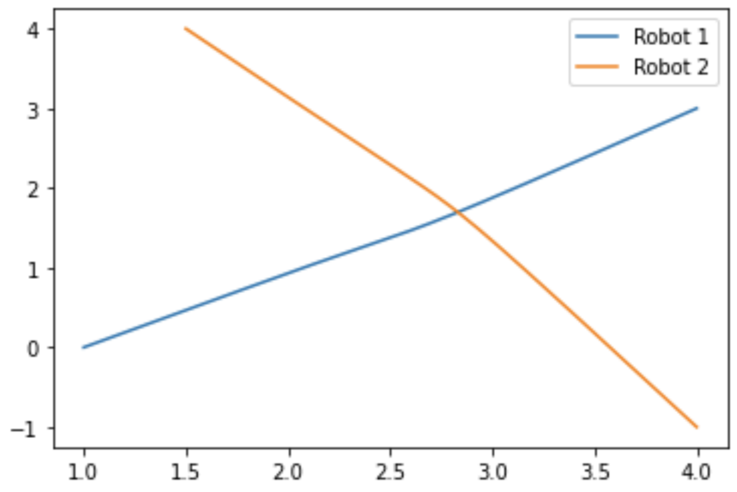


Figure 5.8: Robot 1 with position (1,0) at t = 0 and position (4,3) at t = 1 and robot 2 with position at (4,-1) at t = 0 and position (1.5,4) at t = 1 with **no obstacle**.

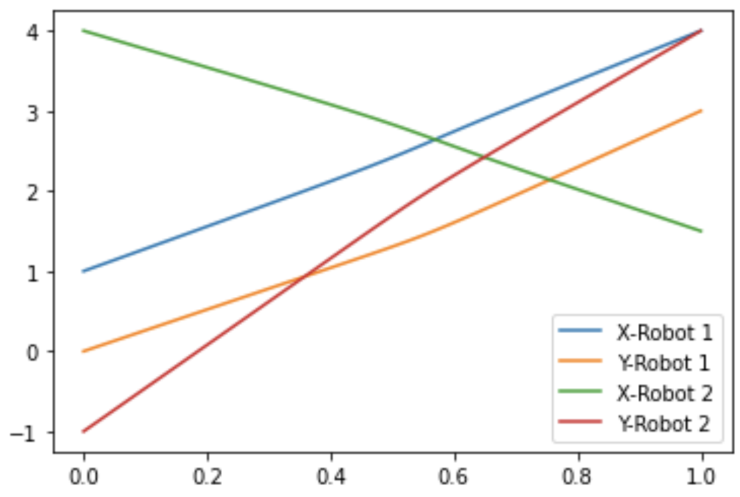


Figure 5.9: Collision free plot

5.1.4 Two Robots And One Obstacle

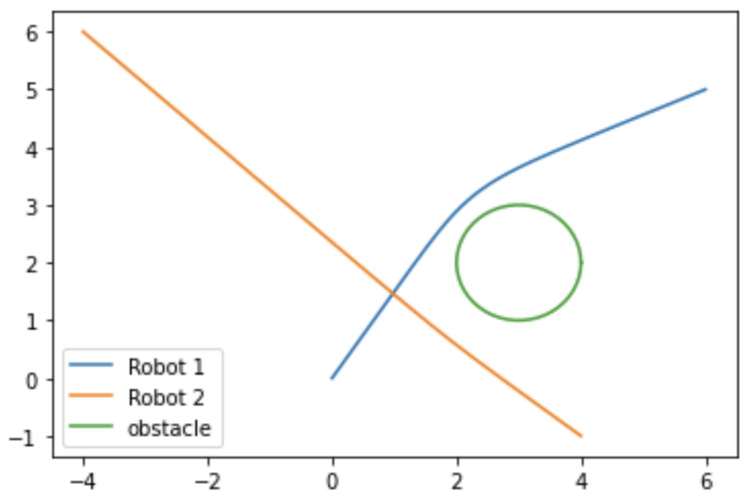


Figure 5.10: Robot 1 with position (0,0) at $t = 0$ and position (4,4) at $t = 1$ and robot 2 with a position at (4,-1) at $t = 0$ and position (0,4) at $t = 1$. Obstacle centered at (2,2) with radius $R = 2$.

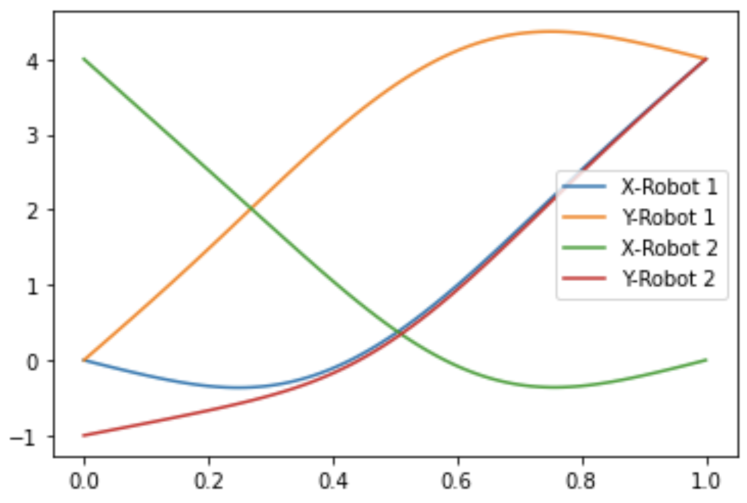


Figure 5.11: Collision free plot of 7

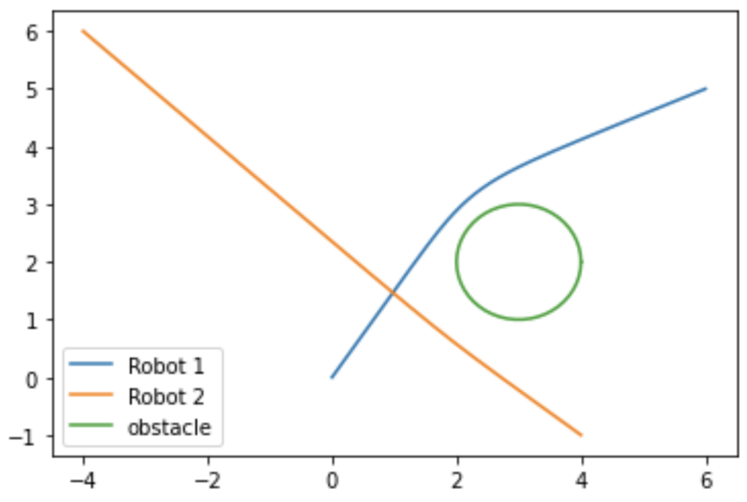


Figure 5.12: Robot 1 with position (0,0) at $t = 0$ and position (6,5) at $t = 1$ and robot 2 with a position at (4,-1) at $t = 0$ and position (-4,6) at $t = 1$. Obstacle centered at (3,2) with radius $R = 1$.

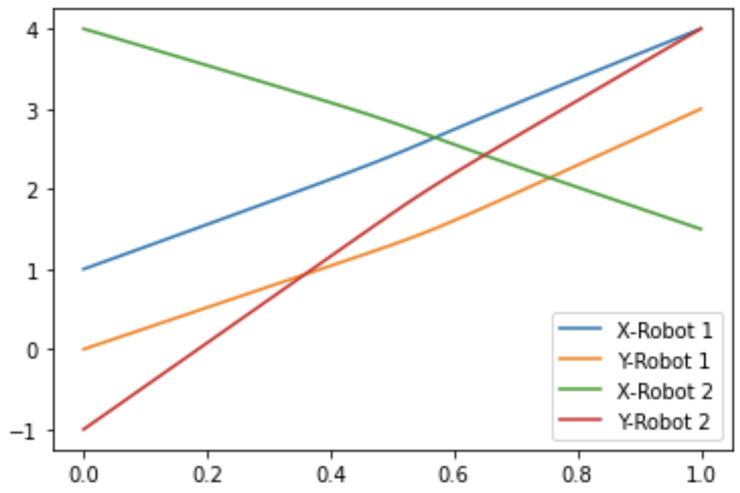


Figure 5.13: Collision free plot of figure 11

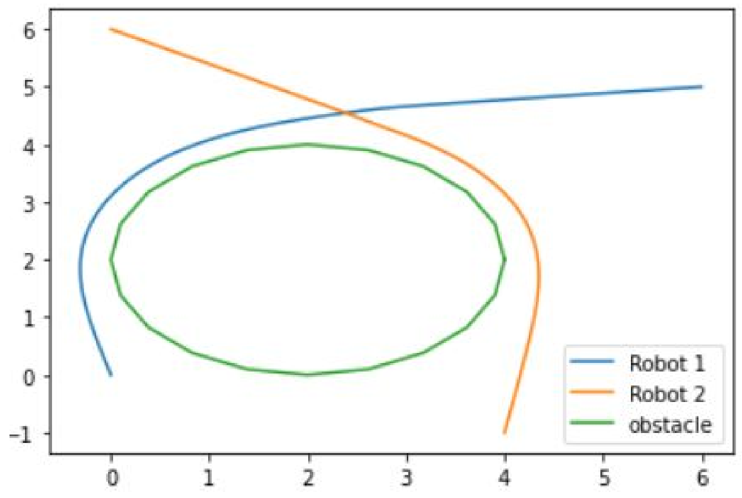


Figure 5.14: Robot 1 with position (-4,6) at $t = 0$ and position (6,5) at $t = 1$ and robot 2 with a position at (-4,6) at $t = 0$ and position (4,-1) at $t = 1$. Obstacle centered at (2,2) with radius $R = 2$.

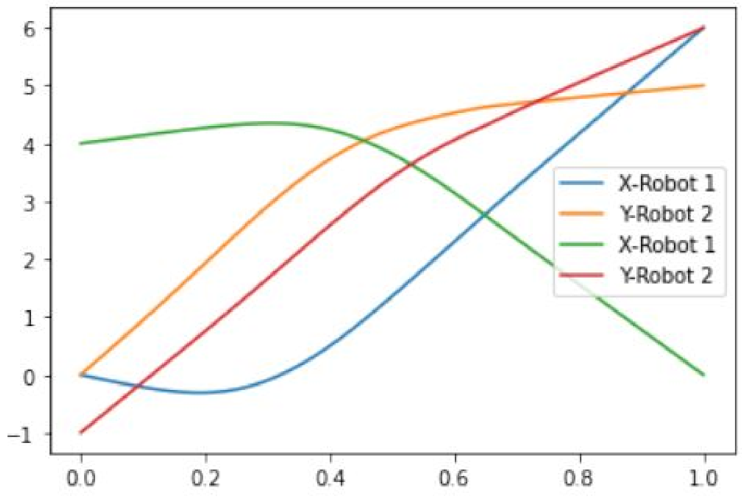
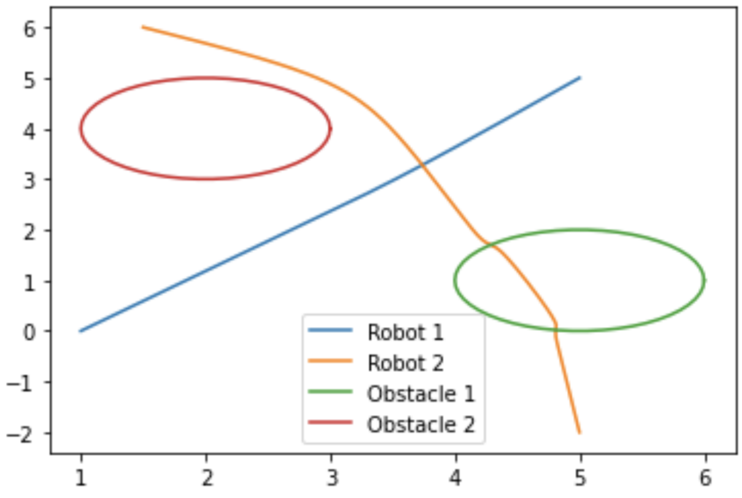


Figure 5.15: Collision free plot of figure 13

5.2 Limitations

First, as we add more robots or obstacles to our model, the algorithm would become increasingly lengthy. For each new robot added, we need to add one term for each existing differential equation and two new differential equations. For each new obstacle added, we need to add one term for each existing differential equation. The result is a boilerplate. Therefore, in the future, we could improve the algorithm by methods like reusing variables, employing automatic code generation, etc.

Second, we did not treat the obstacles in our model as an object that is repulsive in every differential part – rather, it is a hollow ring. That means sometimes, crossing the ring may actually be less costly for the robots (in part due to the deficiency of numerical methods). Hence, if the repulsiveness of the obstacles are not weighted up, the following situation could occur (for case 10.4); however, it can be amended by adjusting weight of repulsiveness of obstacles.



5.3 Summary

To conclude, our model appears to be robust and provides a reasonable approximation of the robots' optimal trajectories. Indeed, the trial cases result in the expected trajectories: in case 1, straight lines are produced, and in all the other cases, the robots never hit each other or the obstacle. Notably, splitting the cost functional into three sub-parts proved advantageous as it allows the weight of the conditions to be manipulated if needed. For example, if desired, one could easily further increase the distance away from the obstacles that the robots are by increasing the weight of the repulsiveness of the obstacles. Indeed, by removing the square root in our length function for the purposes of computation, we increased the prioritization of a shorter path versus not crashing into the obstacles. To offset this, we weighted the repulsiveness higher as well. Overall, the perhaps counter intuitive approach of treating the robots and obstacles as charges seams to be sound.

Given the opportunity for future research, we would seek out a way to program the unsimplified version of our length function (keeping the square root rather than omitting it) in order to produce more reliably accurate results without having to manipulate the weight of each part of the functional. This would require coding our own differential equation solver in Python since the current functions available do not support that type of equation. Another avenue of future exploration could be modifying the problem to have different constraints. For example, rather than just having the robots not crash into each other, we could say that they must maintain a specific minimum distance apart from each other. This would add another condition on the position of the robots beyond just the start and stop boundary conditions. Alternatively, we could also focus on changing the problem to better reflect real life applications. For example, we could extend the model to accommodate 3 dimensions rather than the 2D plane and give the robots a non-negligible volume rather than treating them as point particles. This would necessitate formulating the question as an optimal control problem.

5.4 Contributions

- **Lance Remigio:** Case analysis, plotting, author of LaTex Document.
- **Jin Zhang:** Write algorithm, solving the differential equations numerically.
- **Fatma Djellouli:** Model formulation and first-order necessary conditions derivation.
- **Woody Lee:** Running test cases and construct limitations of the model.