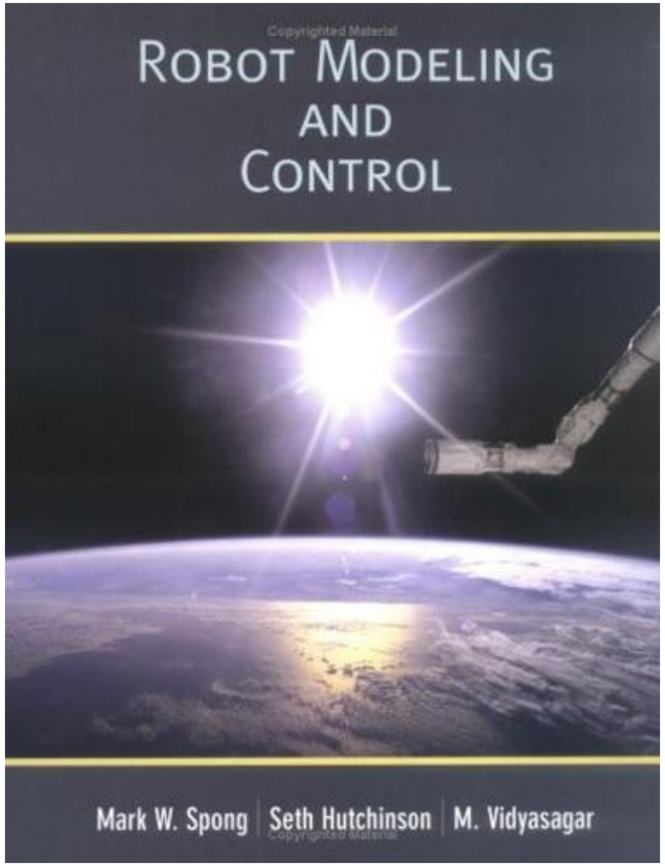
MEAM 520
Lecture 18: Potential Fields

Cynthia Sung, Ph.D.

Mechanical Engineering & Applied Mechanics

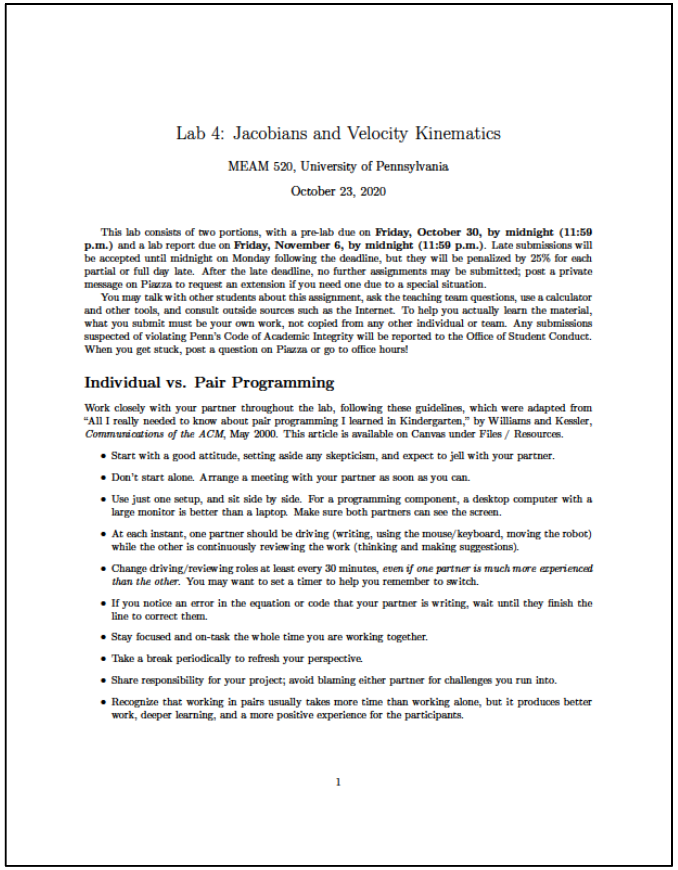
University of Pennsylvania

Today: Trajectory Planning with Potential Fields



Chapter 5: Trajectory Planning

- Read Sec. 5.2



Lab 4: Jacobians and Velocity Kinematics

MEAM 520, University of Pennsylvania

October 23, 2020

This lab consists of two portions, with a pre-lab due on **Friday, October 30, by midnight (11:59 p.m.)** and a lab report due on **Friday, November 6, by midnight (11:59 p.m.)**. Late submissions will be accepted until midnight on Monday following the deadline, but they will be penalized by 25% for each partial or full day late. After the late deadline, no further assignments may be submitted; post a private message on Piazza to request an extension if you need one due to a special situation.

You may talk with other students about this assignment, ask the teaching team questions, use a calculator and other tools, and consult outside sources such as the Internet. To help you actually learn the material, what you submit must be your own work, not copied from any other individual or team. Any submissions suspected of violating Penn's Code of Academic Integrity will be reported to the Office of Student Conduct. When you get stuck, post a question on Piazza or go to office hours!

Individual vs. Pair Programming

Work closely with your partner throughout the lab, following these guidelines, which were adapted from "All I really needed to know about pair programming I learned in Kindergarten," by Williams and Kessler, *Communications of the ACM*, May 2000. This article is available on Canvas under Files / Resources.

- Start with a good attitude, setting aside any skepticism, and expect to jell with your partner.
- Don't start alone. Arrange a meeting with your partner as soon as you can.
- Use just one setup, and sit side by side. For a programming component, a desktop computer with a large monitor is better than a laptop. Make sure both partners can see the screen.
- At each instant, one partner should be driving (writing, using the mouse/keyboard, moving the robot) while the other is continuously reviewing the work (thinking and making suggestions).
- Change driving/reviewing roles at least every 30 minutes, *even if one partner is much more experienced than the other*. You may want to set a timer to help you remember to switch.
- If you notice an error in the equation or code that your partner is writing, wait until they finish the line to correct them.
- Stay focused and on-task the whole time you are working together.
- Take a break periodically to refresh your perspective.
- Share responsibility for your project; avoid blaming either partner for challenges you run into.
- Recognize that working in pairs usually takes more time than working alone, but it produces better work, deeper learning, and a more positive experience for the participants.

1

Lab 4 due Nov. 6

Previously: Manipulator Jacobian

(3 x n) linear velocity Jacobian

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$$

(6 x n) Jacobian
a.k.a. manipulator Jacobian
a.k.a. geometric Jacobian

(3 x n) angular velocity Jacobian

$$J_v(\vec{q}) = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \cdots & \frac{\partial x}{\partial q_n} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \cdots & \frac{\partial y}{\partial q_n} \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \cdots & \frac{\partial z}{\partial q_n} \end{bmatrix}$$

$$J_\omega = [\rho_1 \hat{\mathbf{z}} \quad \rho_2 \mathbf{R}_1^0 \hat{\mathbf{z}} \quad \rho_3 \mathbf{R}_2^0 \hat{\mathbf{z}} \quad \cdots \quad \rho_n \mathbf{R}_{n-1}^0 \hat{\mathbf{z}}]$$

$\rho_i =$ 0 for prismatic
1 for revolute

forward velocity kinematics

$$\xi = J(q)\dot{q}$$

(6 x 1) body velocity

(6 x n) Jacobian

(n x 1) joint velocities

inverse velocity kinematics

$$\dot{q} = J^{-1}\xi$$

Previously: Static Force/Torque Relationships

(n x 1) (n x 6) (6 x 1)

$$\vec{\tau} = J^T(\vec{q})\vec{F}$$

joint forces and torques

Jacobian matrix transpose

endpoint forces and torques

Derivation

$$\vec{\tau}^\top d\vec{q} = \vec{F}^\top d\vec{x}$$

$$d\vec{x} = J_v d\vec{q}$$

$$\vec{\tau}^\top \cancel{d\vec{q}} = \vec{F}^\top J_v \cancel{d\vec{q}}$$

$$\vec{\tau}^\top = \vec{F}^\top J_v$$

$$\vec{\tau} = J_v^\top \vec{F}$$

Simplest to think about for a 3-DOF robot with all revolute joints.
We want to output a force at the tip.

(3 x 1) (3 x 3) (3 x 1)

$$\vec{\tau} = J^T(\vec{q})\vec{F}$$

joint torques

Jacobian matrix transpose

endpoint forces

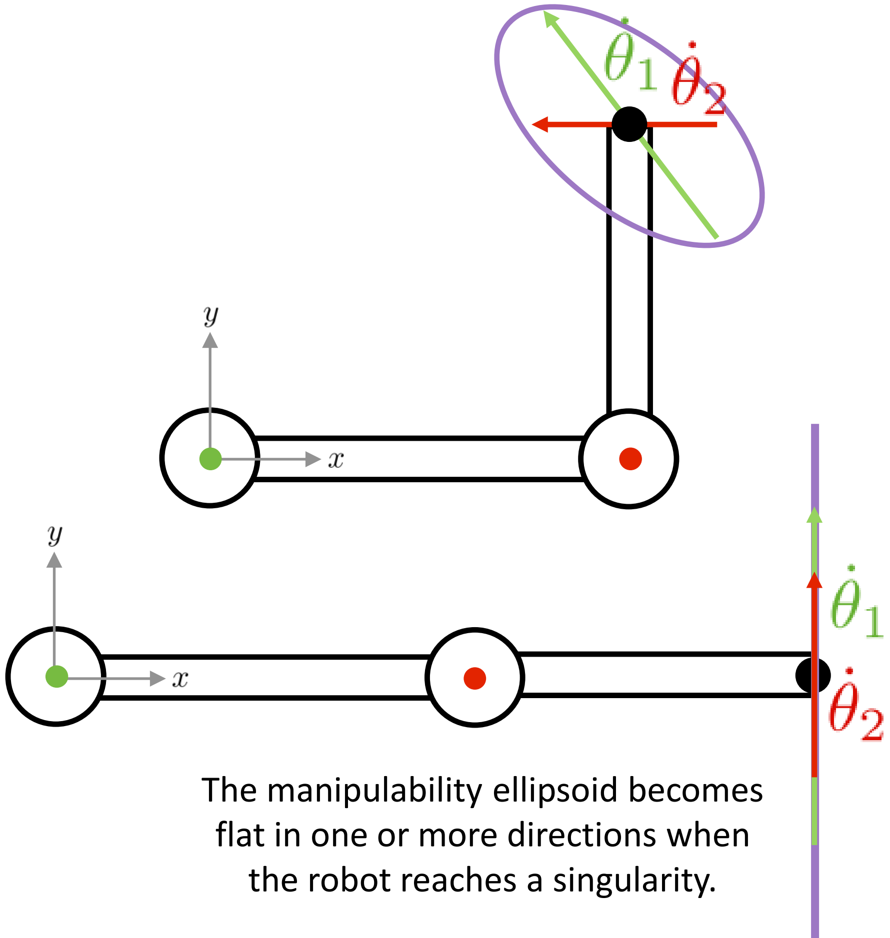
Previously: Manipulability

For a specific configuration, the Jacobian scales the input (joint velocities) to the output (body velocity)

$$\xi = J(q)\dot{q}$$

If you put in a joint velocity vector with unit norm, you can calculate in which direction and how fast the robot's end-effector will translate and rotate.

3D manipulability ellipsoids:
a geometrical representation of all the possible tip velocities (linear or angular) for a normalized joint velocity input.

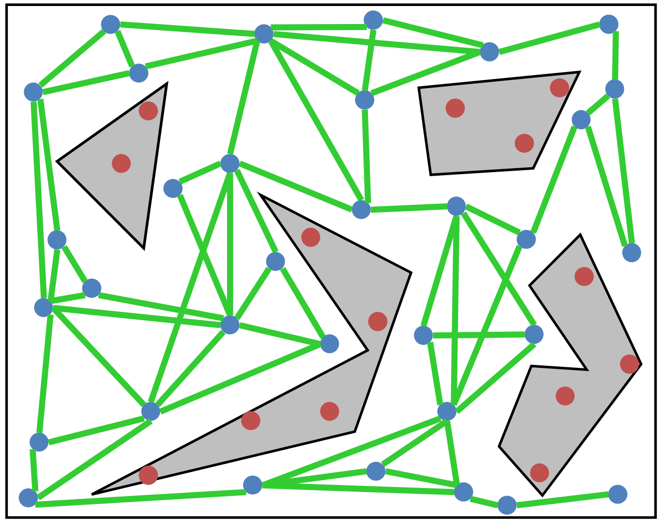


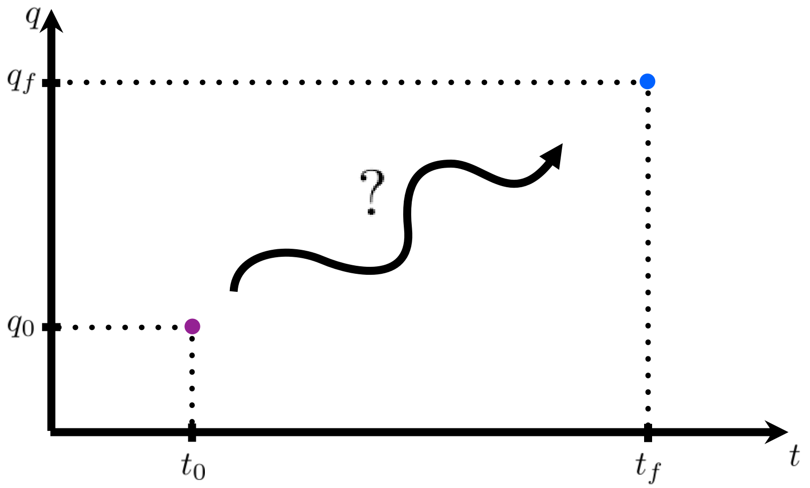
The manipulability ellipsoid becomes flat in one or more directions when the robot reaches a singularity.

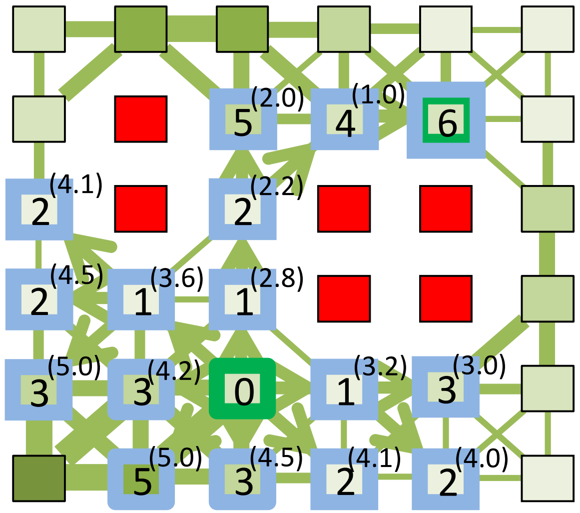
Previously: Path Planning

Planning strategy:

1. Convert your free C-space into a graph/roadmap
2. Find a path from q_{start} to a node q_a that is in the roadmap
3. Find a path from q_{goal} to a node q_b that is in the roadmap
4. Search the roadmap for a path from q_a to q_b







Finding the Free C-Space is Hard

Without requiring an explicit representation of the configuration space obstacle or free configuration space,

find a path from a starting configuration q_s

to a final configuration q_f

such that the robot does not collide with any obstacle as it traverses the path.

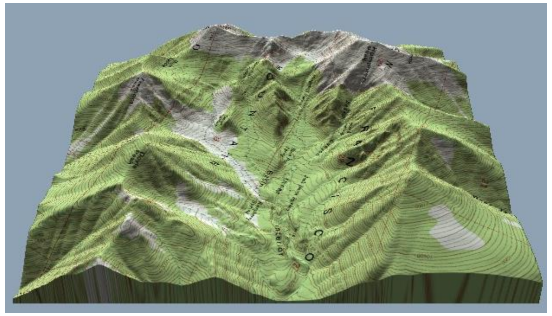
Artificial Potential Fields

Treat the robot as a point particle in the configuration space.

The robot feels forces from an artificial potential field U defined across its configuration space.

We design U to attract the robot to the desired final configuration and repel it from the boundaries of obstacles.

We want one global minimum at goal with no local minima. This is often really difficult to construct!



$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q)$$

Total potential field

Attraction: low potential at the goal

Repulsion: high potential near obstacles

Optimization problem: find the global minimum in U starting from q_s

Use gradient descent

$$\tau(q) = -\nabla U(q)$$

Joint effort

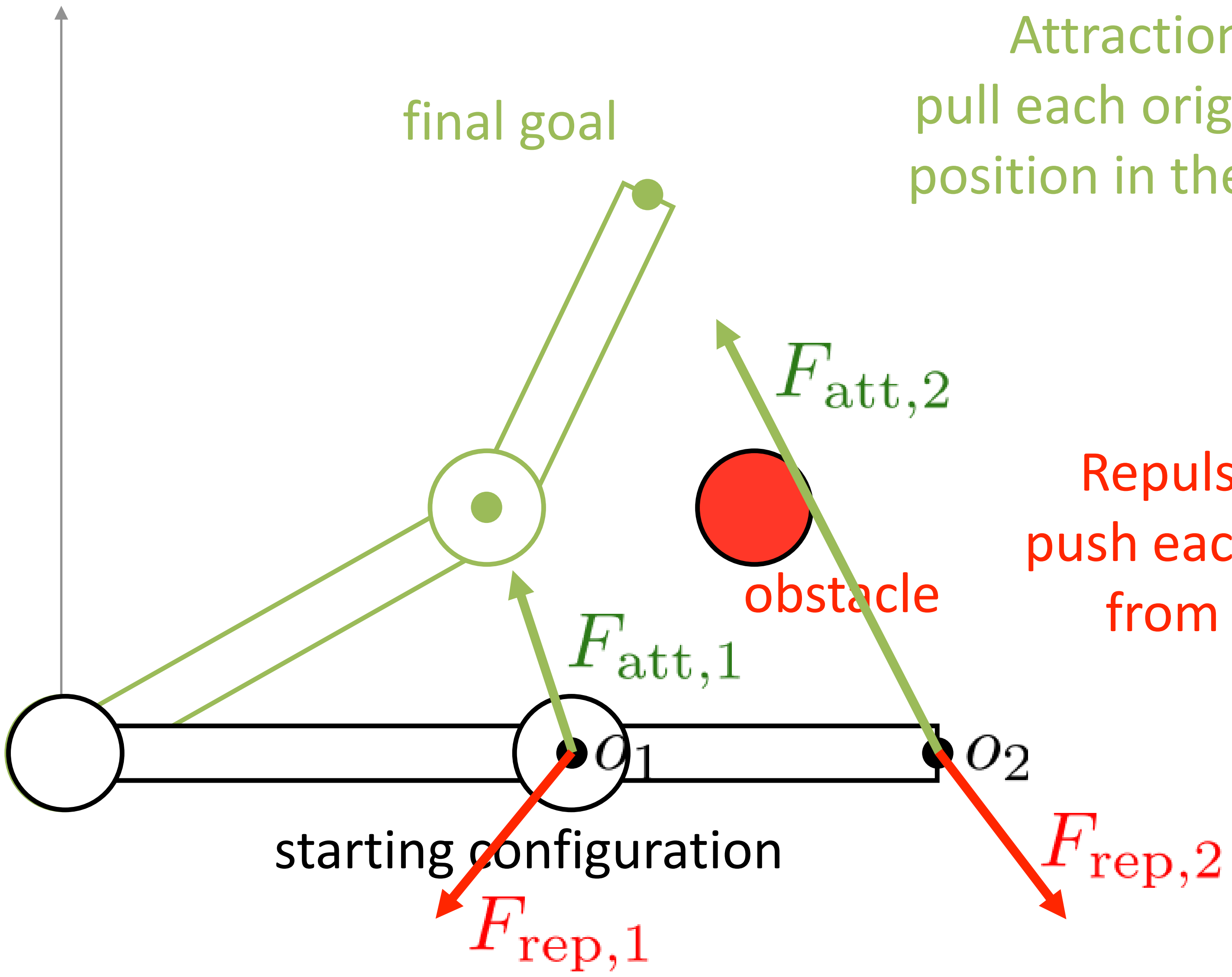
The negative of the gradient of the total potential field: go downhill!

Constructing potential fields directly on the configuration space is difficult because the geometry is complex and you need to know shortest distances to obstacles.

Instead, we define our potential fields directly on the workspace of the robot.

We create a workspace potential field for each DH frame origin (except frame 0) so that it is attracted to its goal location and repelled from obstacles.

Forces on Frame of Origins



Attraction 'forces' pull each origin to its final position in the workspace.

Repulsion 'forces' push each origin away from obstacles.

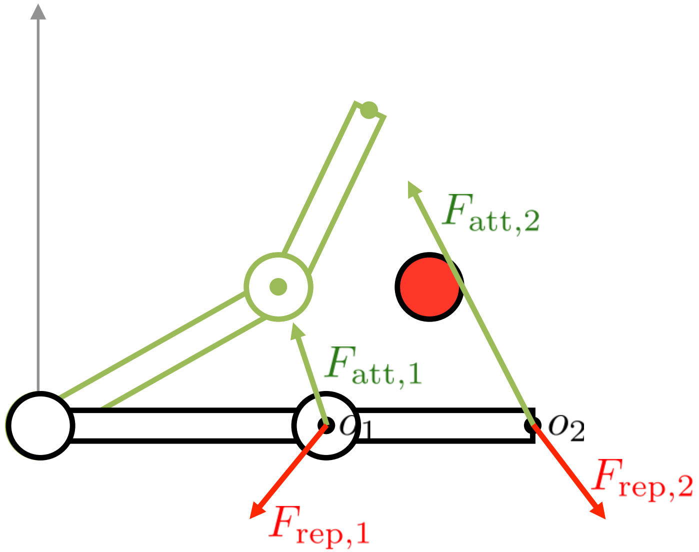
Attractive Field

A simple attractive potential field is the **conic well potential**:

$$U_{\text{att},i}(q) = ||o_i(q) - o_i(q_f)||$$

$$F_{\text{att},i}(q) = -\nabla U_{\text{att},i}(q)$$

$$\boxed{F_{\text{att},i}(q) = -\frac{(o_i(q) - o_i(q_f))}{||(o_i(q) - o_i(q_f))||}}$$



Unit magnitude everywhere, pointing at the final goal. Discontinuity at the goal can cause instability.

Attractive Field

The most common attractive potential field is the **parabolic well potential**:

$$U_{\mathrm{att},i}(q) = \frac{1}{2}\zeta_i ||o_i(q) - o_i(q_f)||^2$$

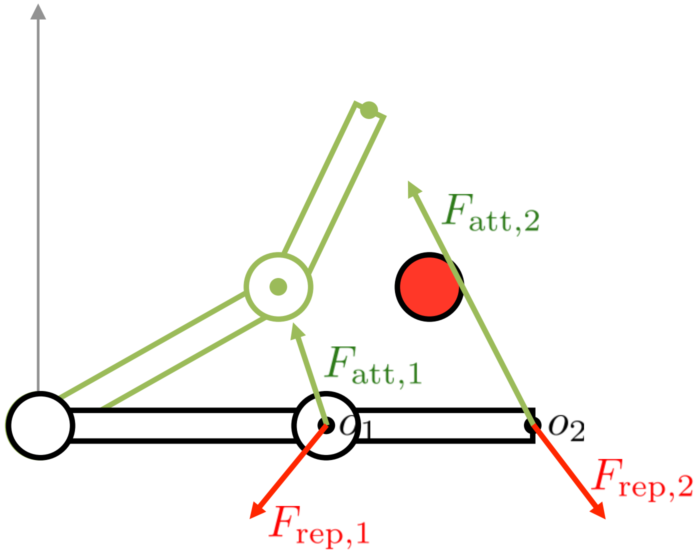
$$F_{\mathrm{att},i}(q) = -\nabla U_{\mathrm{att},i}(q)$$

$$\boxed{F_{\mathrm{att},i}(q) = -\zeta_i \left(o_i(q) - o_i(q_f)\right)}$$

attractive field strength (force per distance)

current position

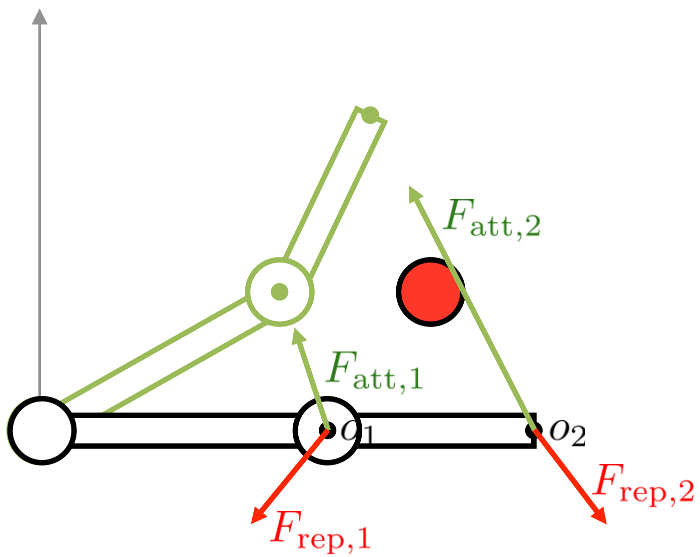
final goal position



Attractive Field

The gradient of the **parabolic well potential**
is very large far from the goal, which
causes very large initial attractive forces.

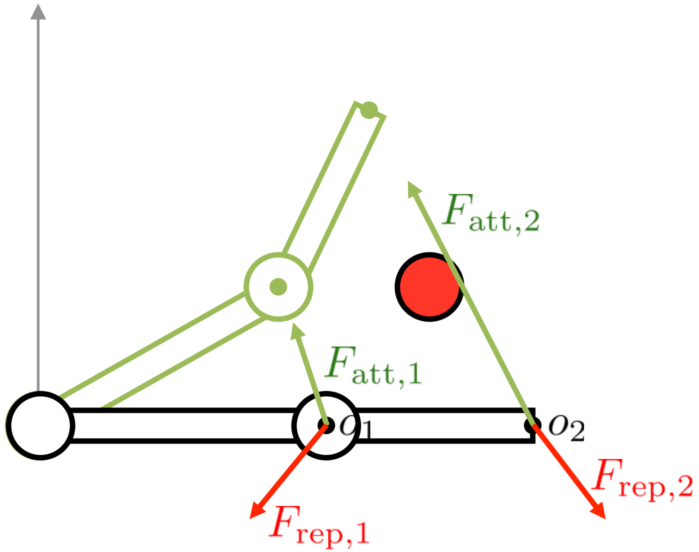
A solution is to use a conic potential far from the goal and
transition (smoothly) to a parabolic
potential closer to the goal.

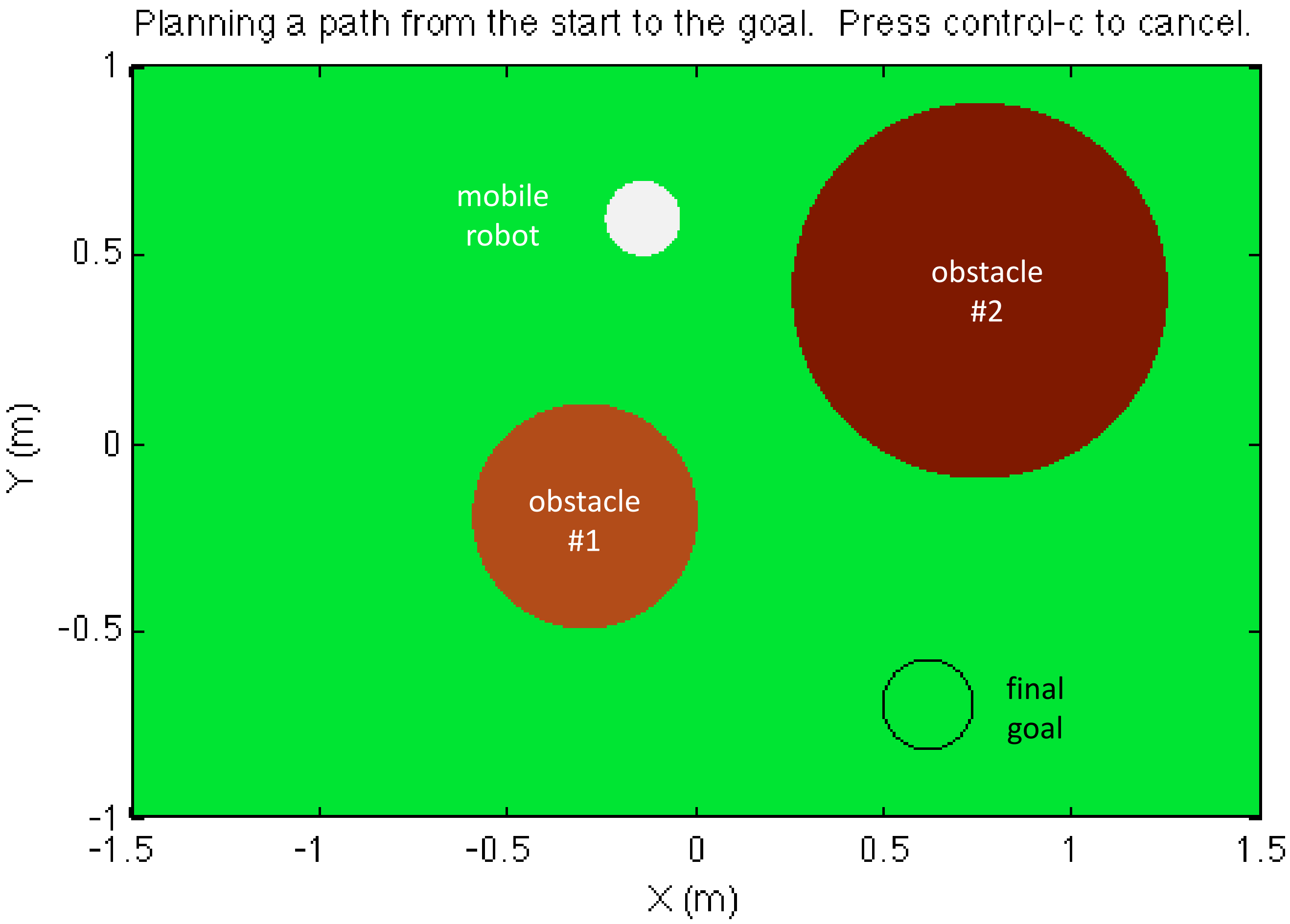


Attractive force has a constant magnitude
when more than a given distance away.

Simple Demo

Mobile robot (PP)
in the plane.



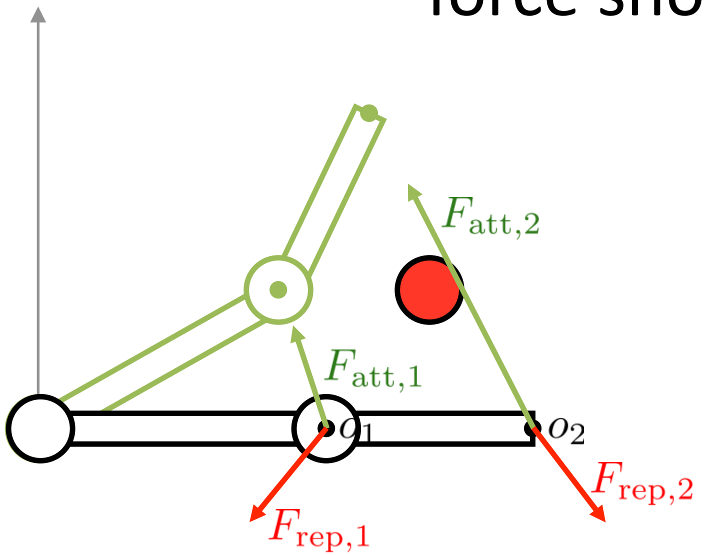


Repulsive Field

We need to prevent collisions between
the robot and all of the obstacles.

Define a **workspace repulsive potential field**
for each frame origin.

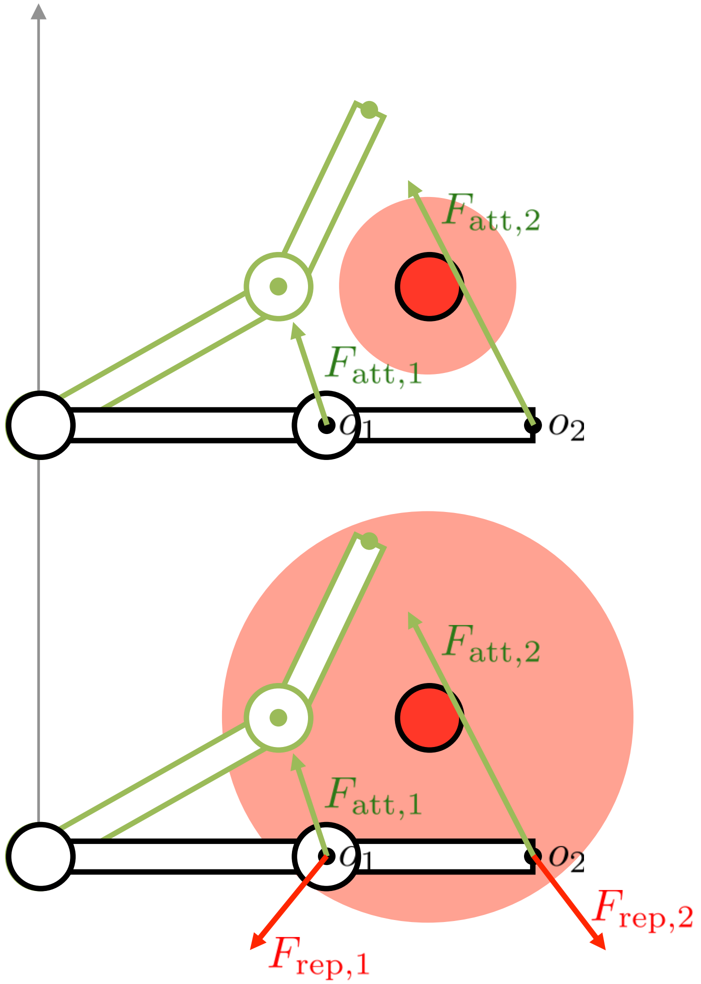
Push very hard on the robot when close:
force should go to infinity to prevent collisions.



Don't push on the robot when it is
farther than a certain distance from
the obstacle.

Repulsive Field

The most common repulsive potential field is as follows:



$$U_{\text{rep},i}(q) = 0 \text{ when } \rho_i(q) > \rho_0$$

shortest distance between oᵢ and the obstacle

distance of influence of the obstacle

$$\text{when } \rho_i(q) \leq \rho_0$$

$$U_{\text{rep},i}(q) = \frac{1}{2}\eta_i \left(\frac{1}{\rho(o_i(q))} - \frac{1}{\rho_0} \right)^2$$

repulsive field strength

Repulsive Field

What is the force on the border of the region of influence?

zero

$$\text{when } \rho_i(q) > \rho_0$$

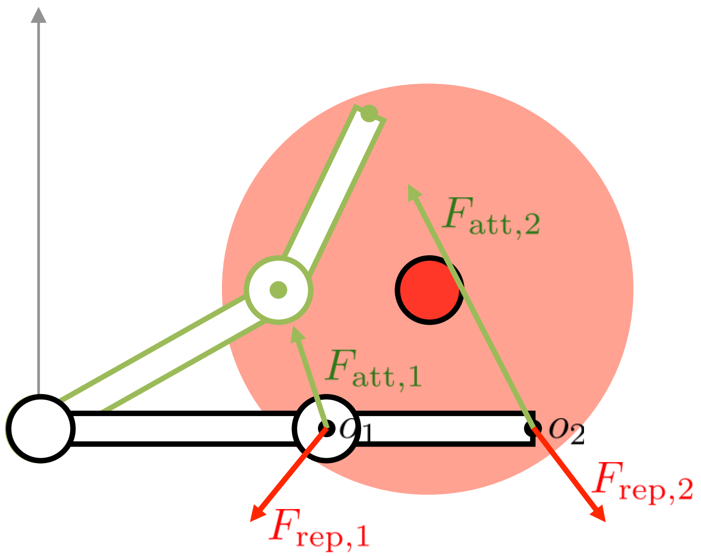
$$\boxed{F_{\text{rep},i}(q) = 0}$$

What is the force on the border of the obstacle?

infinity

$$\text{when } \rho_i(q) \leq \rho_0$$

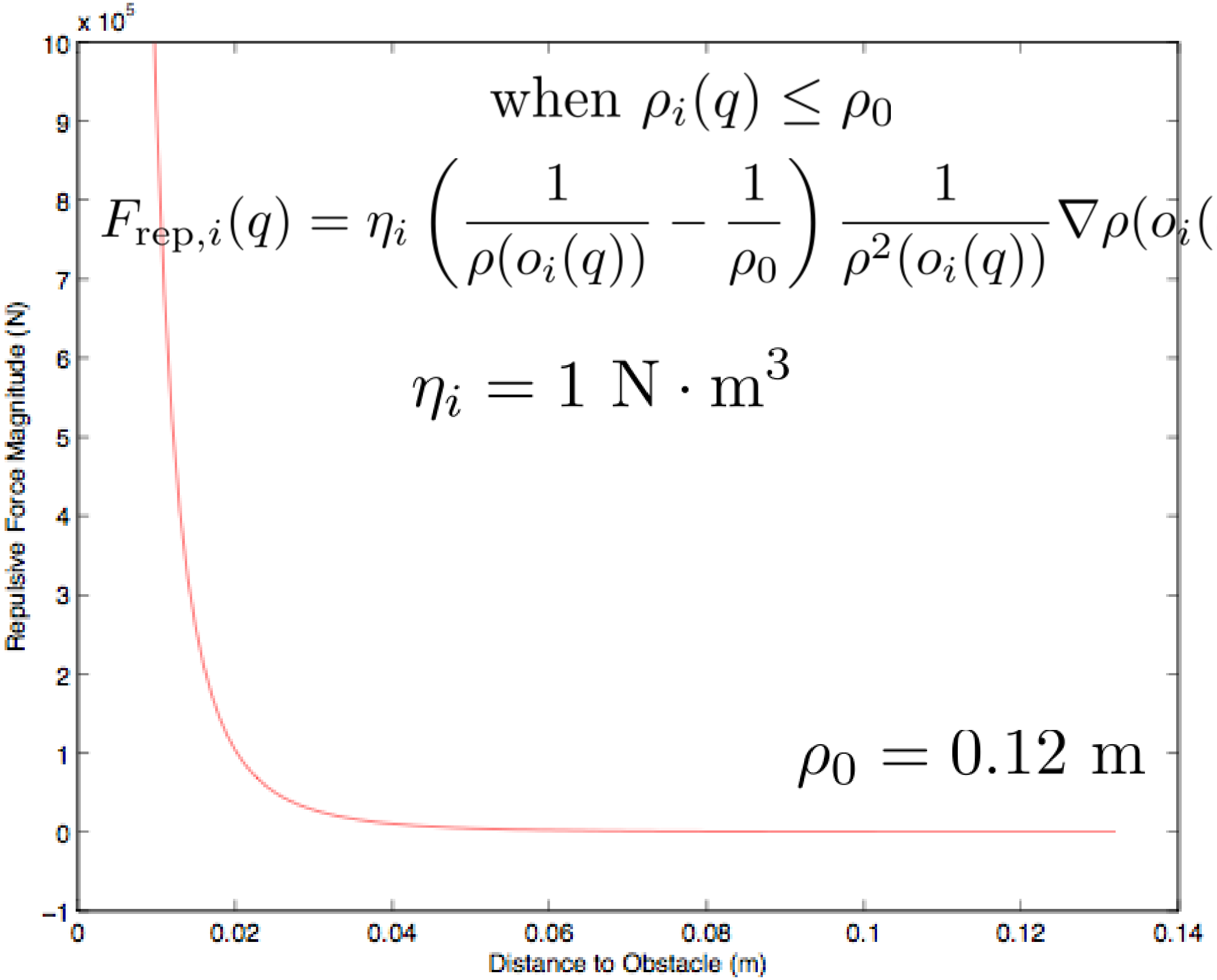
$$\boxed{F_{\text{rep},i}(q) = \eta_i \left(\frac{1}{\rho(o_i(q))} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2(o_i(q))} \nabla \rho(o_i(q))}$$



if the obstacle is convex and
b is the point on obstacle boundary
closest to o_i:

$$\boxed{\nabla \rho(o_i(q)) = \frac{o_i(q) - b}{||o_i(q) - b||}}$$

Repulsive Field



when $\rho_i(q) \leq \rho_0$

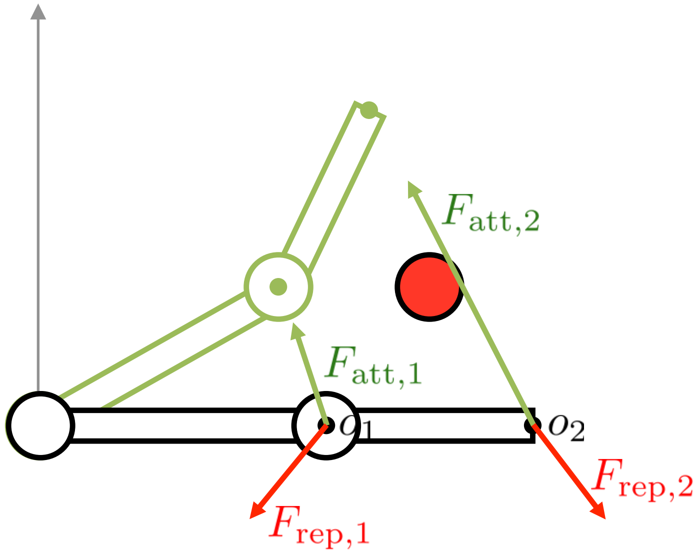
$$F_{\text{rep},i}(q) = \eta_i \left(\frac{1}{\rho(o_i(q))} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2(o_i(q))} \nabla \rho(o_i(q))$$

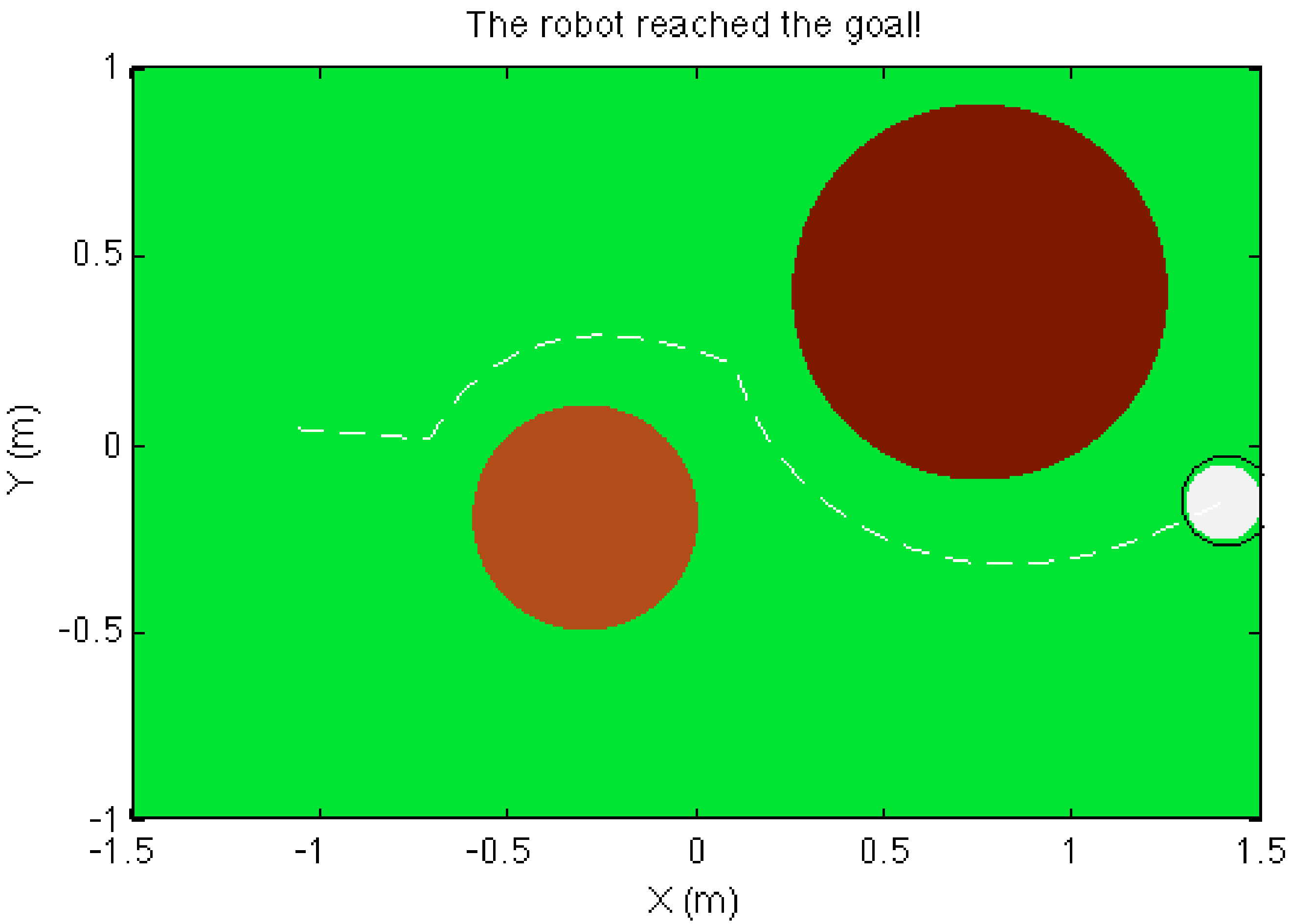
$$\eta_i = 1 \text{ N} \cdot \text{m}^3$$

$$\rho_0 = 0.12 \text{ m}$$

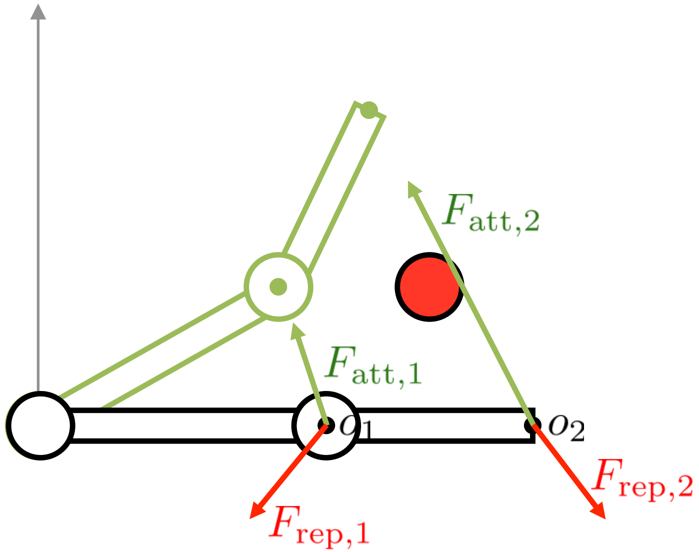
Simple Demo

Mobile robot (PP)
in the plane.





Procedure



while the robot is not at the final goal:

calculate the attractive force on each origin

calculate the repulsive forces on each origin (one force for each obstacle)

convert each workspace force to the equivalent joint-space efforts (torques) using J_v^T

sum all the joint efforts together

take a fixed-magnitude step in joint space in direction of joint efforts to obtain new joint values

check for collisions and goal

Procedure

1. $q^0 \leftarrow q_s, i \leftarrow 0$

2. IF $||q^i - q_f|| > \epsilon$

fixed-size step in joint space

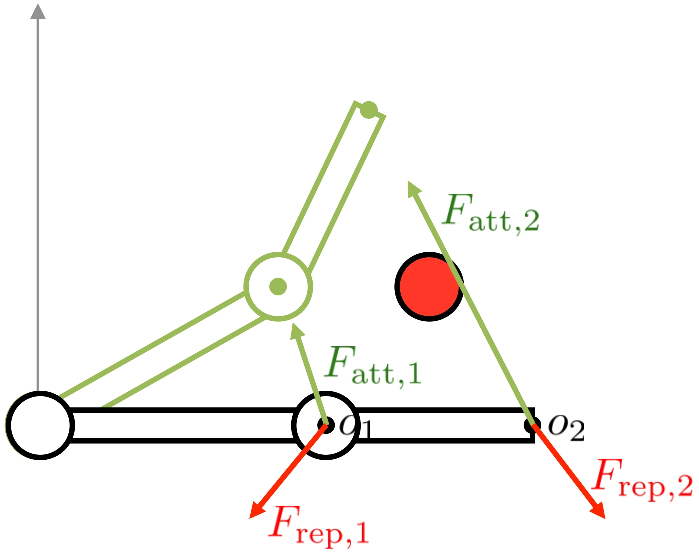
normalized total joint efforts

$$q^{i+1} \leftarrow q^i + \alpha^i \frac{\tau(q^i)}{||\tau(q^i)||}$$

$$i \leftarrow i + 1$$

ELSE return $< q^0, q^1, \ldots, q^i >$

3. GO TO 2



```
% Move the robot while it is more than epsilon from the goal.
while (norm(probot-pf) > epsilon)

    % Calculate the attractive force on the robot.  We are using a
    % parabolic well potential, which always pulls the robot to the
    % goal with a magnitude that is linearly related to the distance
    % between the robot and the final position.
    Fa = -zeta * (probot - pf);
```

Calculate attractive force

```
    % Convert attractive force to joint-space efforts.  In this case,
    % they will be the same because Jv is the identity for a PP robot
    % with orthogonal joints.
    Jv = eye(2);
    taua = Jv'*Fa;
```

Calculate joint efforts needed to create the attractive force

```
    % Calculate the repulsive forces on the robot.

    % Calculate distance between the robot and obstacle 1.
    dobs1 = norm(probot - pobs1) - rrobot - robs1;

    if (dobs1 > rho1)
        % The robot is outside this obstacle's region of influence, so
        % the force is zero.
        Fr1 = [0; 0];
    else
        % The robot is inside this obstacle's region of influence, so
        % the repulsive force must push the robot away.
        Fr1 = eta1 * ((1/dobs1) - (1/rho1)) * (1 / dobs1^2) * ...
            (probot - pobs1)/norm(probot-pobs1);
    end
```

Calculate repulsive force from obstacle 1

```
    % Calculate distance between the robot and obstacle 2.
    dobs2 = norm(probot - pobs2) - rrobot - robs2;

    if (dobs2 > rho2)
```

Calculate repulsive force from obstacle 2

```
        % The robot is outside this obstacle's region of influence, so
        % the force is zero.
        Fr2 = [0; 0];
    else
        % The robot is inside this obstacle's region of influence, so
        % the repulsive force must push the robot away.
        Fr2 = eta2 * ((1/dobs2) - (1/rho2)) * (1 / dobs2^2) * (probot - p
    end

    % Convert repulsive force to joint-space efforts.  In this case,
    % they will be the same because Jv is the identity for a PP robot
    % with orthogonal joints.
    taur1 = Jv'*Fr1;
    taur2 = Jv'*Fr2;
```

Calculate joint efforts needed to create the two repulsive forces

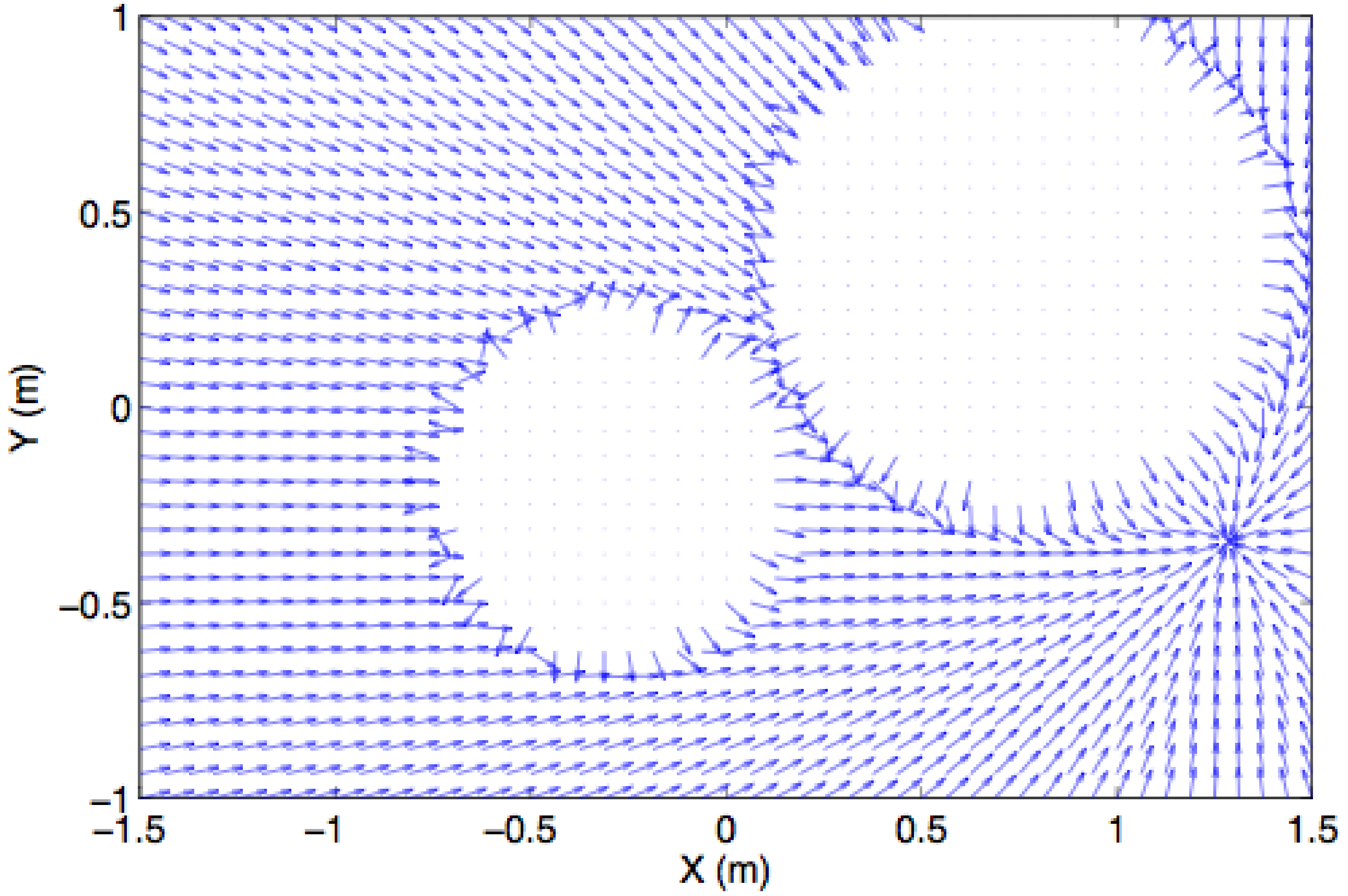
```
    % Sum the torques due to the attractive and repulsive forces
    % together. This is actually forces for our robot, but we are
    % following SHV naming conventions.
    tau = taua + taur1 + taur2;
```

Sum all the joint efforts together

```
    % Calculate the change in position as a scaled version of the net
    % torque.
    probot = probot + alpha * tau / norm(tau);
```

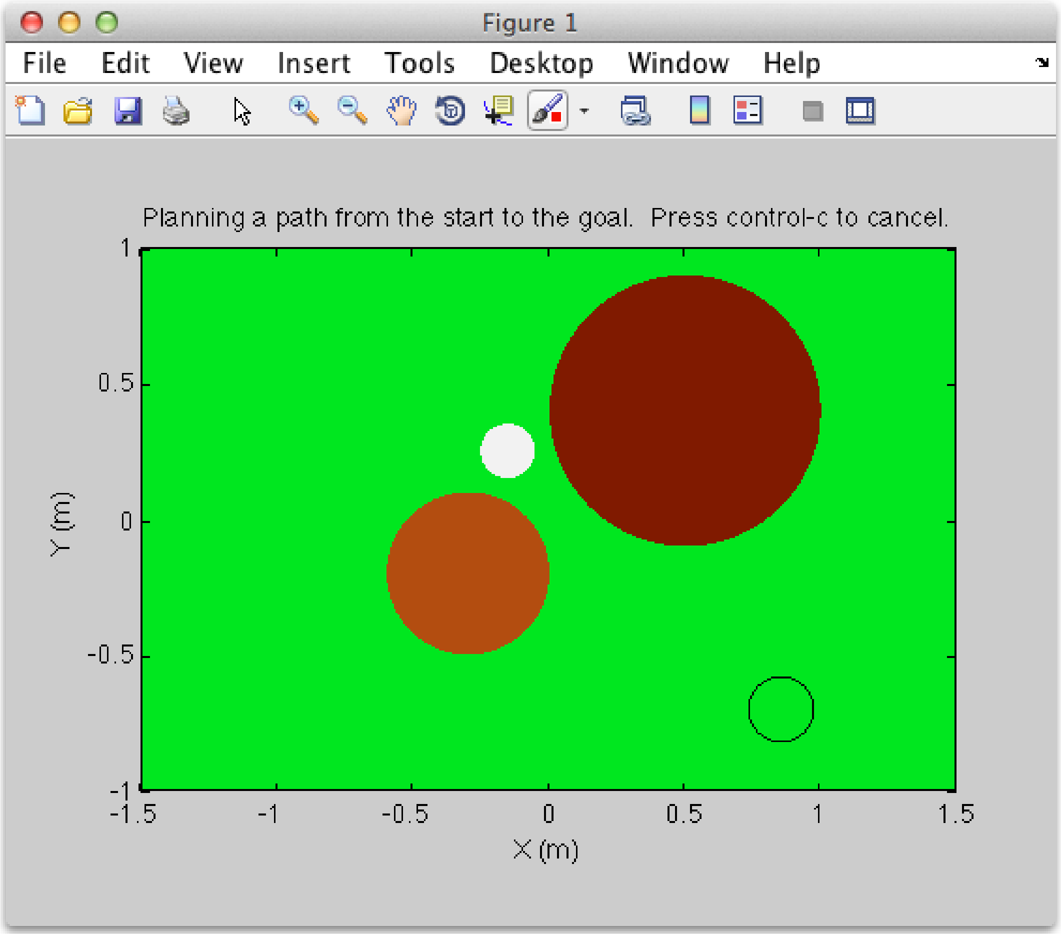
Calculate a new pose for the robot exactly alpha away in joint space in the direction of the summed joint effort.

```
    % Store this robot position.
    probothistory(:,end+1) = probot;
```



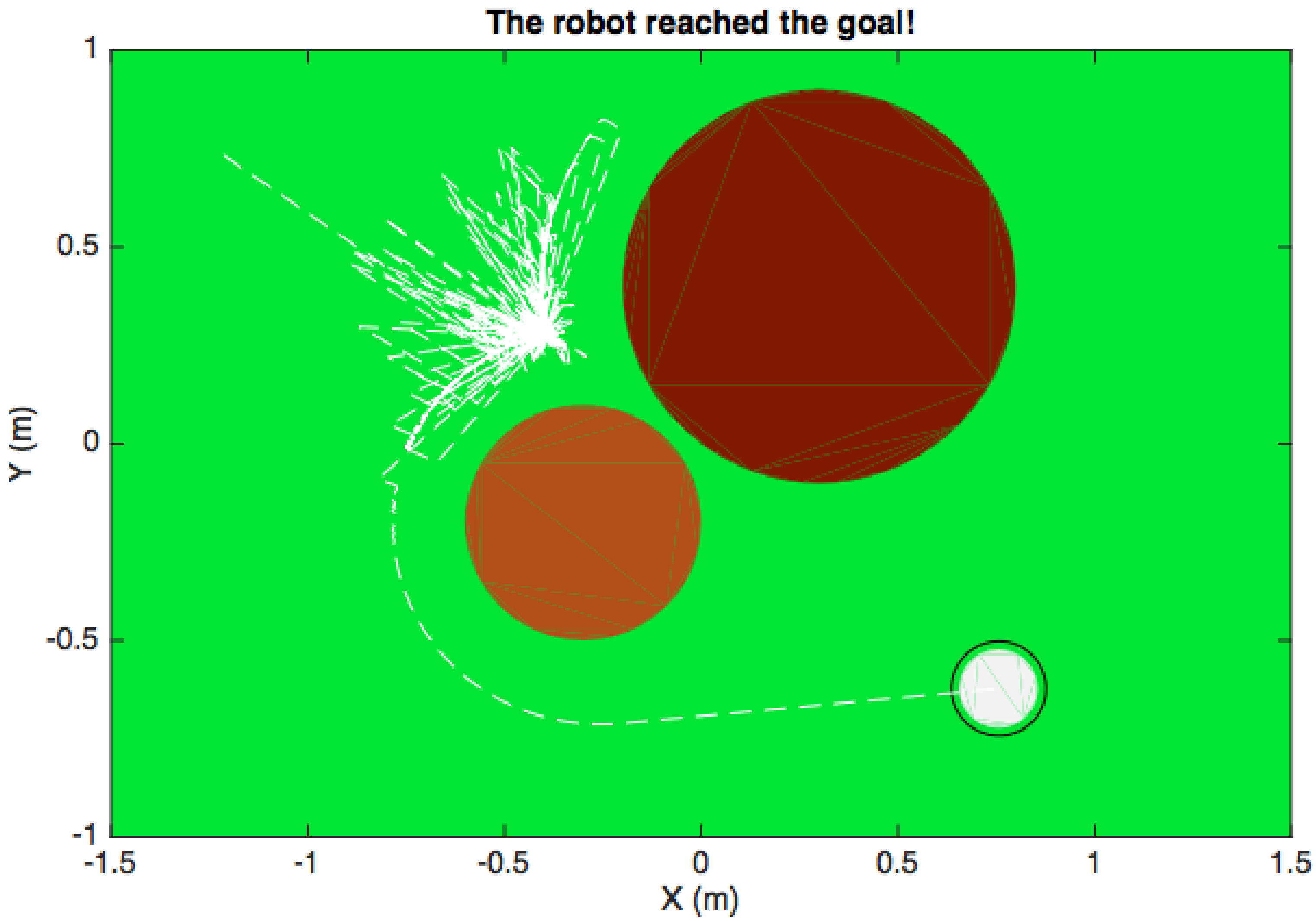
1
0.5
0
−0.5
−1
Y (m)
−1.5
−1
−0.5
0
0.5
1
1.5
X (m)

The robot can get stuck at a local minimum.

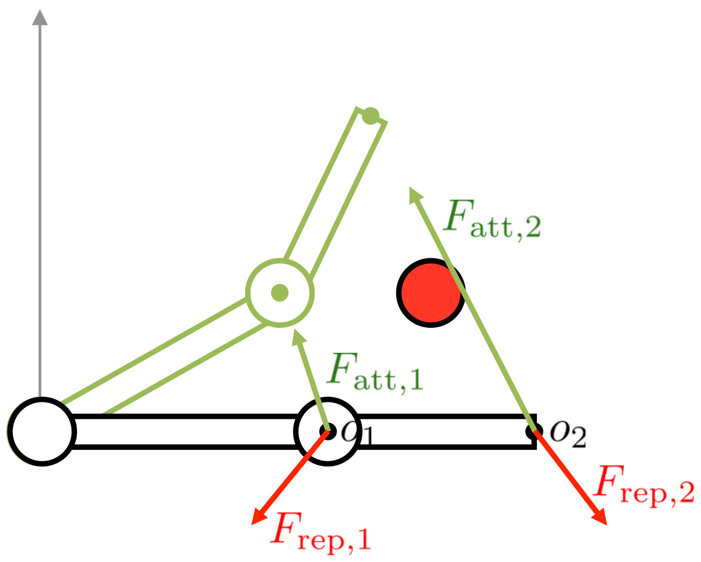


SHV 5.3 explains how to escape local minima: detect lack of motion and do a random walk.

```matlab
    % Check for local minima if requested.
    if (checkForLocalMinima)

        % Get the size of the robot's history; we can't check three
        % steps back in the robot's history if it doesn't yet have
        % four steps.
        s = size(probothistory);

        % If there are more than four elements in the robot history.
        if (s(2) > 4)
            % Check the current position against each of the last
            % three positions, and compare the distance to the
            % threshold epsilonm.
            if (((norm(probot-probothistory(:,end-1)) < epsilonm) && ...
                    ((norm(probot-probothistory(:,end-2)) < epsilonm))) && ...
                    (norm(probot-probothistory(:,end-3)) < epsilonm))
                % The robot is probably at a local minimum.
                disp('The robot is at a local minimum.  Do a random walk.')

                % Remember where the robot is now.
                localmin(1) = probot(1);
                localmin(2) = probot(2);

                % Move the robot to the middle of one of the
                % obstacles so that we can choose a new non-colliding
                % position as often as is needed.
                probot(1) = xobs1;
                probot(2) = yobs1;

                % Check if the proposed new position is inside either
                % of the obstacles.  If it is, this is not a good
                % position, so we should pick a new one.
                while ((norm(pobs1 - probot) < (rrobot + robs1)) || ...
                        (norm(pobs2 - probot) < (rrobot + robs2)))
                    % Perturb the robot's position by a fixed amount
                    % in a random direction.  This is like a one-step
                    % random walk, but in a random direction rather
                    % than always positive or negative on each
                    % coordinate.
                    theta = 2*pi*rand(1);
                    probot(1) = localmin(1) + v * cos(theta);
                    probot(2) = localmin(2) + v * sin(theta);
                end

                % Store this new position in the robobot's history.
                probothistory(:,end+1) = probot;

                % Increase the size of the random walk over time to
                % try to escape bigger local minima.
                v = v * 1.02;
            end
        end
    end
```



The robot reached the goal!
1
0.5
0
-0.5
-1
Y (m)
-1.5
-1
-0.5
0
0.5
1
1.5
X (m)



How do we apply the attractive and repulsive forces to a robotic arm?

You need to formulate the linear velocity Jacobian *for each origin*.

$$\vec{\tau}_1 = J_{v,1}^{\top} \vec{F}_1 \qquad \vec{\tau}_2 = J_{v,2}^{\top} \vec{F}_2$$

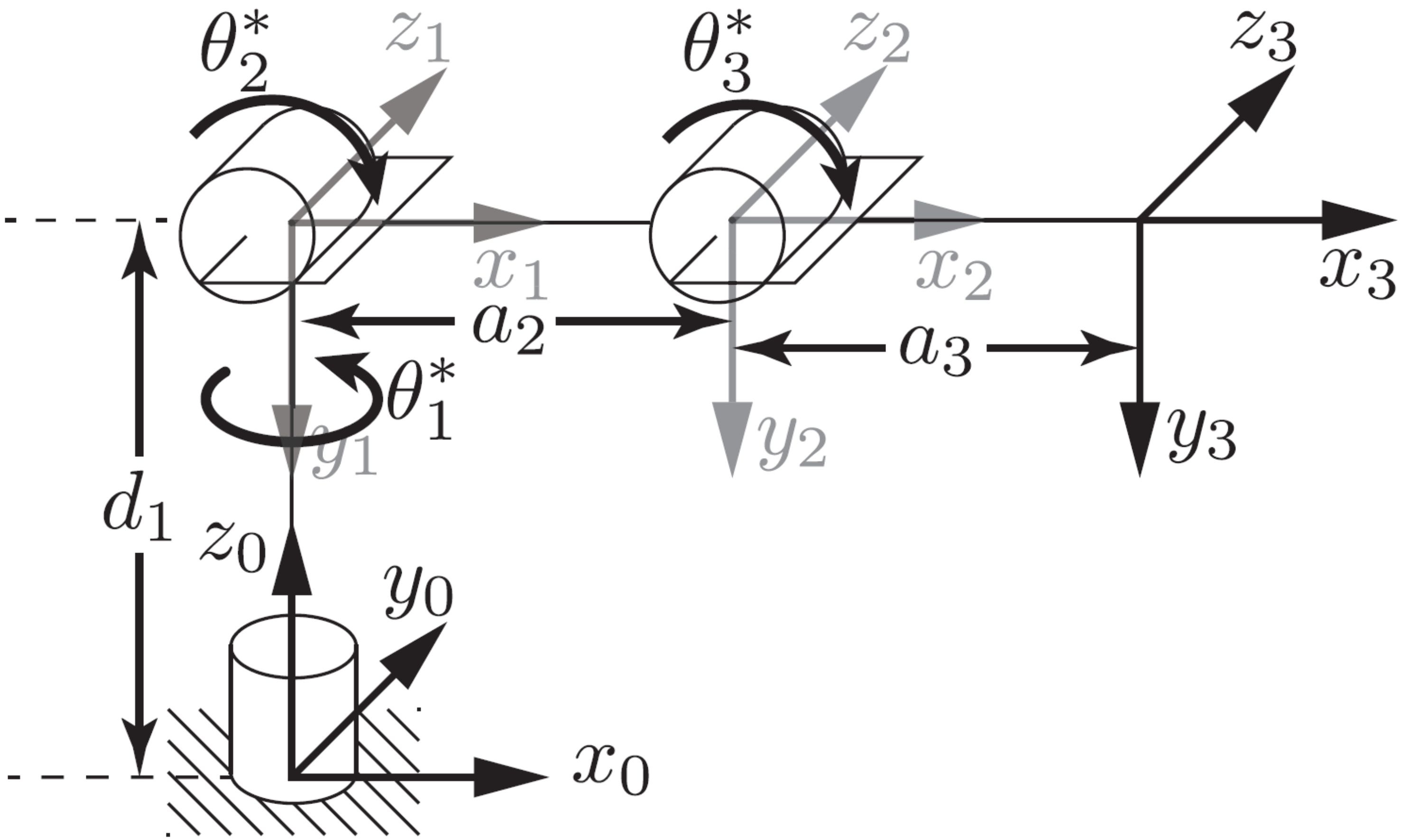
Columns for joints after that origin will be zeros.

Sum the 'forces' on that origin and determine the joint velocities.

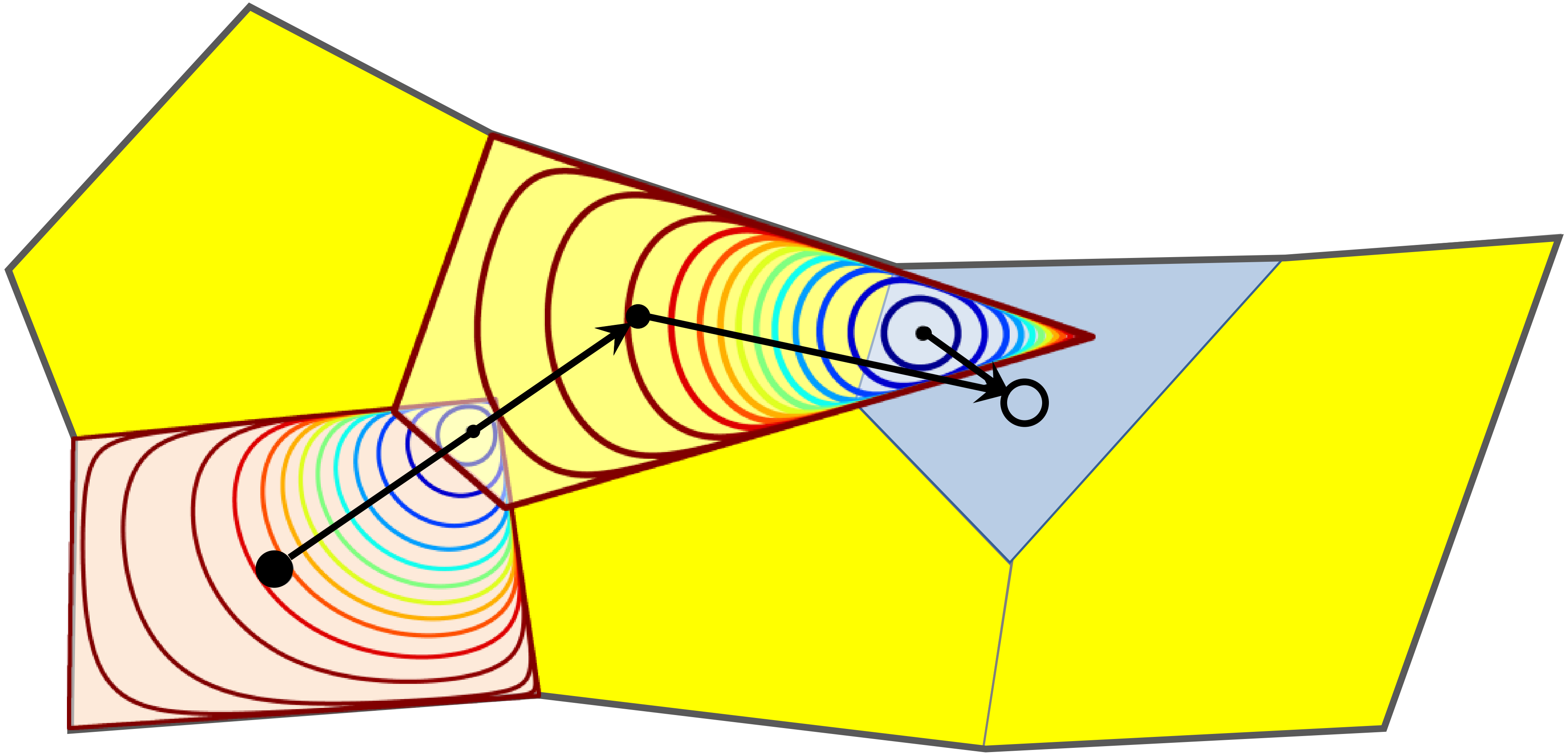
Note that this approach may still allow collisions along the links: you would need to add a floating repulsion point to guarantee no collisions.

This is not necessary for Lab 5

Example: Articulated RRR Arm



Combining planning methods



Ayanian, Nora, Vinutha Kallem, and Vijay Kumar. "Synthesis of feedback controllers for multiple aerial robots with geometric constraints." *IROS 2011.*

Next Time: Paper Reading

2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

FrE4.2

Grasping POMDPs

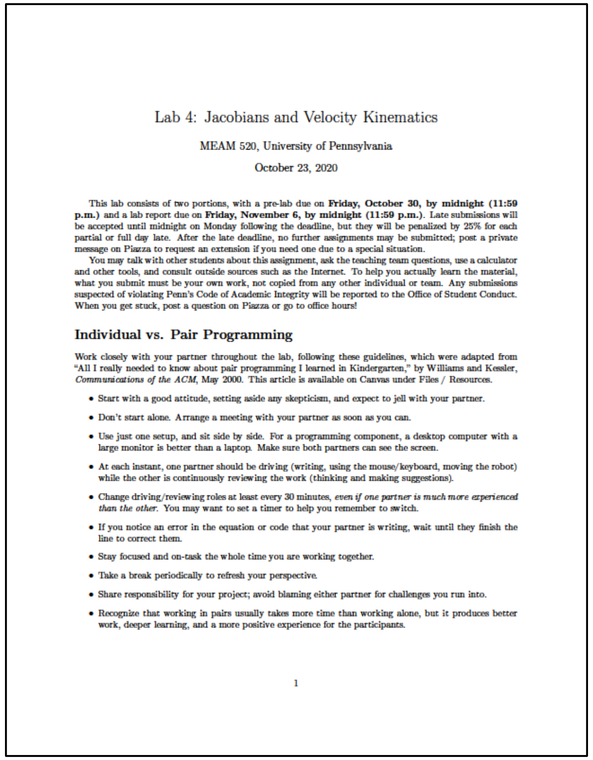
Kaijen Hsiao and Leslie Pack Kaelbling and Tomás Lozano-Pérez

I. Introduction

II. Background and Approach

1-4244-0602-1/07/$20.00 ©2007 IEEE. 4685

Hsiao, K and Kaelbling, LP, and Lozano-Pérez, T. “Grasping POMDPs.” ICRA 2007.



Lab 4: Jacobians and Velocity Kinematics

MEAM 520, University of Pennsylvania

October 23, 2020

Individual vs. Pair Programming

1

Lab 4 due Nov. 6