COMP 417: Homework 1 Solutions

October 2020

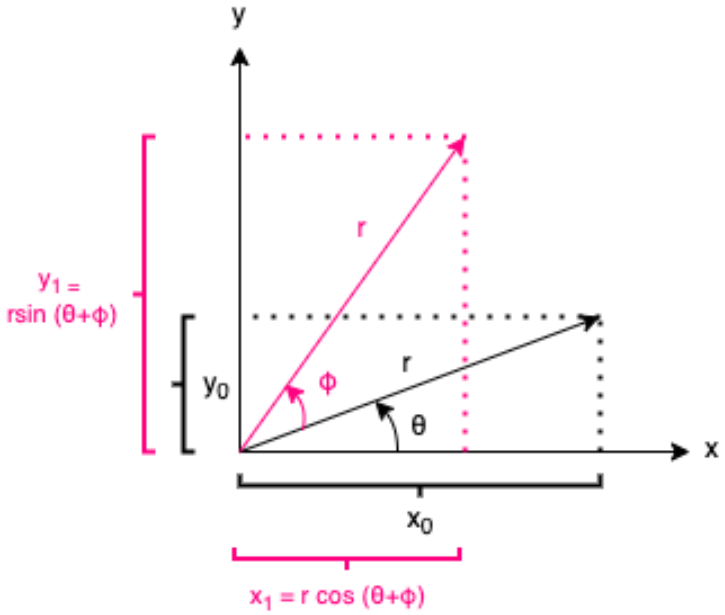

Figure 1: Showing the Cartesian coordinate with a point rotated around ICR (0,0).

Q1: 2D Rotation Transform (3 points)

1. $x_1 = r\cos(\theta+\phi) = r(\cos\theta\cos\phi - \sin\theta\sin\phi) = \underbrace{r\cos\theta}_{x_0}\cos\phi - \underbrace{r\sin\theta}_{y_0}\sin\phi = \boxed{x_0\cos\phi - y_0\sin\phi}$

2. $y_1 = r\sin(\theta+\phi) = r(\sin\theta\cos\phi + \cos\theta\sin\phi) = \underbrace{r\sin\theta}_{y_0}\cos\phi + \underbrace{r\cos\theta}_{x_0}\sin\phi = \boxed{x_0\sin\phi + y_0\cos\phi}$

Using equations 1 and 2 we can drive a matrix form as follows:

$$\begin{bmatrix} x_1 \\ y_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \omega\delta t \end{bmatrix} = \begin{bmatrix} x_0\cos\phi - y_0\sin\phi \\ x_0\sin\phi + y_0\cos\phi \\ \theta+\phi \end{bmatrix}$$

Q2: Forward and Inverse Kinematics (4 points) This question is intended to give you practice facing some of the modeling tasks common when you pick up a new robot. There are always a few choices that can make sense, so we will not just be looking for a single answer, but rather that you connected all the pieces together in a sensible way based on the assumptions you made. Specifically, a few things that would all be about equivalent (some harder and some easier for sure!):

- Treating time as a discrete step, Δt, or as continuous dt. The benefit of using continuous time, is we can always check for the moment we might hit a constraint and just ensure we don't make that motion. For Δt solutions, you need to say something about the whole path that might be taken over the duration (such as a check if the angle θ_2 would wrap-around.
- The "zero-point" of the angles is suggested to be when the arm is straight up, but the diagram was a bit confusing for some people due to Dave's poor ability to draw in MS Word. It's OK if you ignored the written assumption and used angle values as if $(\theta_1, \theta_2) = (\pi/2, \pi)$ when the arm is straight up. Then of course, you have to adjust the joint angle limits to avoid collisions with yourself and the table.
- Collision-checking and returning false/fail from the different routines. There is no standard way to set this up in robotics worldwide, so we won't be picky. But, we want to see that you mentioned something about this in your descriptions or function code.

State Space: $[x, y] \in \mathcal{R}^2$. Using the rule that "the state captures everything we need to know to do the task", we can simply use the x,y position of the hand, following the stated assumption about the nature of tasks. However, it's totally OK to add joint angles to this (necessary for the definition of kinematics in differential form). If you propose that the state would be **only joint angles**, then you must say something about how to relate these to the hand's position to accomplish goals.

Action Space: Joints articulations $[\theta_1, \theta_2] \in \mathcal{R}^2$. Can also be joint velocities in differential form. Could potentially be x,y velocities, but then should say something about how to achieve these.

Forward Kinematics The Forward Kinematics function takes a state and action as the input and calculates the resulting state. There are a wide variety of solutions here depending on the choices that have been made previously. The pseudo-code or explanation should mainly describe the forward geometry that maps joint angles (or discrete changes or continuous changes of the same) to end effector position or velocity. Here you would mostly just derive the forward equations of motion and paste them inside your function to return the result. An example derivation is shown in Figure 2.

- fkine(angle of each joint) $\rightarrow$ position of the end-effector

Inverse Kinematics The Inverse Kinematics function/algorithm takes a target state as the input, and calculates the action required. In our final-angle action representation, there is a simple solution using polar coordinates:

- Compute $r = \sqrt{x^2 + y^2}$, $\phi = atan(y/x)$, the polar coordinate representation of the goal point around the global frame.
- Note that θ_2 controls the r of the hand because it "bends" the arm, changing its effective radius in the range from 2 (fully extended) to 0 (tucked completed in half). Make an equilateral triangle from the two links with length one. Its interior angle, θ_i relates to radius like $sin(theta_i/2) = r/2$. We did not use θ_i exactly to represent θ_2, so we need to first solve for $theta_i = 2 * asin(r/2)$, then do a bit of trig to get $\theta_2 = \pi - 2 * asin(r/2)$.

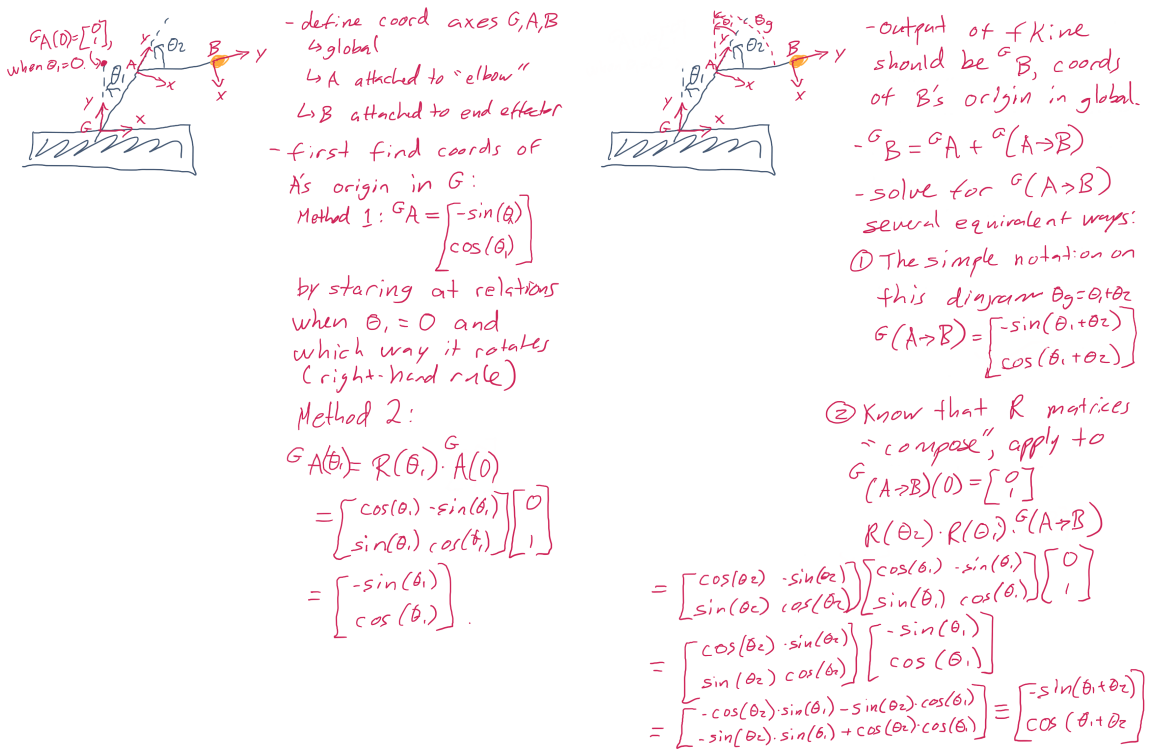

Figure 2: An example derivation of the math relating joint angles to end-effector position. Simply outputting the values of the hand in the global coordinate frame is a reasonable answer for forward kinematics in the Δt representation. Taking derivatives of these equations is one way to get to the dt version.

- Note that if we control θ_2 in this way, then θ_1 is free to put the hand at the correct ϕ. We can again use our equilateral triangle to find something related to θ_2, ϕ and θ_1 sum to $\pi/2$ (the signs and trig details are not very important, but you can get them with care). Finally $\theta_1 = \phi - \pi/2 - \theta_2/2$ works well.

We need to make a few key notes that are more important than the precise equation details. There are two *good* solutions in many cases: the elbow can be left or right, but the hand will still be in the same place. This is due to the fact that we have not defined a target rotation. For some spots, such as the hand being near the table, we need to rule out one of the solutions that puts the elbow into the table. This is easy to do by symmetry: for positive θ_1, we'll prefer positive θ_2 and likewise negative and more negative. In robotics we call this "elbow up" inverse kinematics and it is a real thing used on industrial arms mounted on tables like the one we see here. You can guess what elbow mode makes most sense for arms hanging from the ceiling!

Note that only goals that live within the top half circle of radius 2 centred at (0,0) are reasonable goals, so we need to return fail for things that have a radius outside of $[0, 2]$ or a ϕ outside of $[0, \pi]$. For anything inside this circle, you should be able to return a solution, just so long as you're careful not to accidentally find a solution that puts the elbow down. We'll hope you mentioned collisions and resolving multiple solutions somehow, but it may be very different for differential solutions and other choices you made.

Q3: Choose the Algorithm (5 points)

1. The Bug 1 Algorithm (with right-hand rule)? **No**, it doesn't follow the right-hand rule because it turns

the wrong way at the first obstacle. Also is not simply a Bug 1 with left-hand rule, because the path turns the opposite way later on.

2. The Bug 2 Algorithm (with right-hand rule)? **No**, it doesn't follow the right-hand rule. Also is not simply a Bug 2 with left-hand rule, because the path turns the opposite way later on.

3. Dijkstra's algorithm? **Yes**, possible when graph formed with 8-neighbor motion model. Looks like the shortest possible path under those assumptions, which Dijkstra's always finds. If other assumptions are stated, we will judge those based on proper application of Dijkstra's logic.

4. The A* algorithm? **Yes**, A* finds the same final paths as Dijkstra's. The only difference is that A* opens the nodes in a prioritized order by using a heuristic function. Since we are only looking at the final path returned, we can't tell them apart, so we'll check that you answer the same for both.

5. The RRT algorithm? **No**, it is vanishingly unlikely that a basic RRT finds the shortest path.