Daniel Pickem, Andrew Melim, Jarius Tillman, Matheus Svolenski

RIP - Robot Intelligence - Planning CS 4649/7649 Fall 2013 Assignment 2 11/25/2013

Deliverables include the following:

- Email the Instructor AND TA a PDF summary that describes your work, results and answers to all the questions posed below. Thoroughness in analysis, plots, great movies and answers in the reports are the primary component of your grade!
- A zip file that includes your code in an organized form
- A README file describing how to run the code to re-create the results (DO NOT send movies).
- A one page summary describing what each group member contributed.
- Print the summary and bring it to class on November 26th as well as a USB stick with movies.

Problem 1 - Jacobian Control

Implementing Jacobian control using forward and inverse kinematics of a 3 DoF robot arm was relatively straightforward. At each iteration the algorithm computes the following quantities:

- Compute the inverse of the current Jacobian matrix.
- Compute the joint velocities using the inverse Jacobian matrix.
- Update the joint positions using the joint velocities and the previously known joint positions.
- Recompute the Jacobian matrix using the new joint positions.
- Compute the new end effector position and the position error between end effector and target position of end effector.
- Compute target directed end effector velocities.

We noticed a caveat of this approach during implementation. The initial and target configuration are given in terms of configuration space. This is not a problem for the target configuration, but the robot has to be initialized with joint angles. The reason for that is because the Jacobian control approach requires the computation of the Jacobian matrix, which requires the joint angles as initialization parameters. To compute the joint angles corresponding to the initial configuration of $[2.6, 1.3, 1.0]$, we initialized the robot with random joint angles and used our controller to direct the robot to the initial position and read out the joint angles at that position.

Problem 2 - RRTs

The implementations of the following four RRT variants used a few parameters including the *number of samples*, the *step size*, the *goal probability*, and an *error threshold*. We will elaborate on their influence on the algorithms' behavior in the following discussion.

- **Baseline:** This version of RRTs used only the number of samples and the step size, where the step size essentially dictates how fast and dense the configuration space is covered. A small step size results in a relatively dense tree that covers a smaller volume whereas a large step size results in a sparse tree that covers a larger volume.

The following implementations were applied to a different environment with more obstacles. Therefore, the conclusions and observations made should stand separately from the base case.

- **Goal-directed:** This variant of RRTs moves towards the goal with a certain probability p and towards a random configuration otherwise. Therefore, the parameters that influence this algorithm are the *goal probability*, the *error threshold*, and very importantly the *step size*. We noticed that if the step size is set to too large values, the robot arm tends to "jump over" obstacles without triggering the collision detection. Therefore, it is important to set the step size (which essentially represents a dt for updating the joint angles $q = q + \dot{q}dt$) to small values, i.e. such that $\max\{\dot{q}\}\ dt\ <$ smallest obstacle. Comparing those two quantities requires either applying forward kinematics to the joint velocities to compute the end effector velocities or computing the obstacle dimensions in configuration space. In our algorithm we used forward kinematics to estimate the largest step size. In addition, we used the step size to estimate the error threshold, which we set such that the arm could not "jump over" the target configuration.
  The probability of moving towards the goal was selected to be very small, around $p = 0.01$, so in only one percent of the cases does the tree expand towards the goal. Depending on the complexity of the environment and the number of obstacles this resembled the following *connect* RRT version.

- **Connect:** This version of RRTs attempts to expand towards the goal for as many consecutive steps as possible (i.e. until a collision is detected). The behavior of this algorithm was very similar to the goal-directed version and the reason is the following. In an environment with many obstacles (as was the case in *file2.scene*) a randomly sampled node is likely to be either in an obstacle or cause the arm to collide with an obstacle with high probability. On the other hand an expansion towards the goal, if possible, enables many more subsequent expansions towards the goal. Therefore, once a goal expansion was successful, subsequent goal expansions are very likely even though the goal probability is low.

- **Bidirectional:** This variant of RRTs grows two trees, one rooted at the initial and one at the target configuration. While this has the potential to speed up computation of a path we observed that it can take as long to connect these two trees as it takes to grow one tree from initial to target configuration. In addition, while the goal-directed and connect versions of RRT generally produce a smooth approach of the target configuration, the bidirectional version approached the target in a rather jerky fashion.

Problem 3 - Task Constraints

To approach the problem of adding constraints to the robot planner, we implemented a hard constraint for the arm motion along the y-axis in the workspace. This type of constraint is useful if the arm is manipulating objects which require specific orientations or motions.

We implemented the First-Order retraction method which directly minimizes the constraint error of the sampled point. Each iteration, we compute the error of the end-effector in workspace from the sampled RRT point in joint space from forward kinematic equations. Next, we compute the Jacobian locally around the current sampled joint configuration and then update the sampled point by a small motion towards a local minima. The minima that we converge to is dependent upon the configuration the Jacobian is computed from since it's only valid in a small, local region.

In our experiments, we noticed that the inital configuration of the arm is not suitable for motion to the provided goal. During planning, we noticed that our implementation deviates from the starting point and even the constraint a bit to reconfigure the joint orientation. Once reconfigured, the end-effector then follows along the constrained path. This was quite surprising, yet makes intuitive sense. The planner noticed that it could not directly find a set of actions which satisfied the constraints, so it first modified it's configuration and then executed the plan. Note however, that the action is a bit noisy and imprecise. This could be solved with the use of a path smoother and some higher level logic to separate reconfiguration and action execution we believe.

Problem 4 - Questions

(a)

For collision checking, a function would be required that would allow possible expansion nodes to be rejected if there is a collision between the links and other objects in workspace. This would require knowing the current joint positions in workspace and using the Jacobian to detect what the next orientation will be given a change in joint velocity. If a collision is detected, that node should not be considered in the RRT; however, if there is not a collision the node can be added.

For joint limits, there should be a constraint placed on the possible positions that each link can have with respect to each other. This is easiest done in configuration space vs. workspace. By taking the configuration space and choosing what are the maximum and minimum angles between links,

Table 1: Averaged results of three RRT variants: Goal-directed, connected, and bidirectional RRTs.

RRT-Variant	**# of nodes**	**Time [sec]**
Connect	358.4	131.25
Goal-directed	388.5	143.50
Bidirectional	1341	8.67

it should be possible to create functions (either continuous or piecewise) that restrict the possible movements of certain joints given the current orientation of a specific one. This can then be applied into the RRT in configuration space directly using a similar method to the collision node rejection function.

(b)

In terms of computation of time and number of nodes, the trees resulting from the connected and directed RRT versions were very similar. Both times were a lot slower than the bidirectional which is expected due to the extra logic used in determining which nodes to add to the graph. The bidirectional tree runs a lot faster than the goal-directed and connected trees because it does not use any biasing logic when it chooses the next node to add to the tree. The goal-directed tree would often get stuck hitting the obstacle in the lower right, compared to the connected algorithm which would mostly stay close to the large obstacle. Unfortunately, this resulted in the connected arm sometimes repeatedly hitting the large central obstacle due to the "move towards goal until collision" logic. In the lower middle part of the path, the goal-directed would often times create a wider sweep than the connected version, in this region the connected region would usually take a much more direct path due to the lack of obstacles in that region. Compared to goal-directed and connected the bi-directional tree is more than double the number of nodes, which might be expected when joining two trees. Even though the two trees are most likely to meet up in the lower mid area of the environment, the rate at which they reach this point and their graph size and shape is less determined due to the lack of node choice biasing. This also affects the final path of the arm which is simply a joining of the path created between the random shared node on each tree. In this central area, there was a lot of jerky movements from the arm due to the jagged path that connected the shared node to the roots of both trees. All paths had instances of jerkiness in their paths which is common in RRT when path smoothing is not added to the algorithm but it was most often visible in the bidirectional tree.

(c)

In the task constrained problem, the only limitation for the joints is that the cup is maintained in an upright position. Unfortunately, this constraint does not say anything more about how the links are positioned with respect to each other as long as this objective is maintained. It may be that

Table 2: Raw Results of three RRT variants: Goal-directed, connected, and bidirectional RRTs.

RRT-Variant	**# of nodes**	**Time [sec]**
Connect	245	96.89
	458	159.27
	414	136.17
	368	152.52
	254	91.02
	379	131.69
	327	113.23
	409	157.03
	438	161.76
	292	112.89
Goal-directed	356	133.67
	423	176.02
	313	119.23
	361	149.39
	427	175.37
	398	134.81
	515	216.71
	329	114.72
	481	135.45
	282	79.63
Bidirectional	555	3.41
	1159	7.26
	1517	9.79
	794	5.01
	2998	19.18
	527	3.52
	1489	9.54
	1241	7.85
	2239	14.53
	891	6.00

the final joint orientation is such that the cup's opening is blocked or that the cup is enclosed by the links, which would make access to it difficult or impossible. This can be fixed by using a joint limitation function similar to that in Part a). It may also be possible to create pseudo-objects in the environment (i.e modify the robot's perception of the cup such that the opening is always given a berth of a specific distance or radius).

Additionally, it's very important to verify that the initial configuration of the arm will work for the entire motion. Having to reconfigure the orientation of joints and the end-effector midway through a motion while holding a cup would be extremely undesirable.

Summary

This section summarizes the contribution of each team member.

1. Matheus Svolenski
   - Helped evaluate algorithms for problem 4b.
2. Jarius Tillman
   - Answered questions of problem 4.
   - Tested and evaluated algorithms for problem 4b.
   - Writeup for section 3 in the document.
3. Andrew Melim
   - Implemented task-constrained control for problem 3.
   - Created videos and animations for problem 3.
   - Writeup for section 3 in the document.
4. Daniel Pickem
   - Implemented Jacobian control approach using inverse and forward kinematics for problem 1.
   - Implemented the four RRT versions required for problem 2.
   - Created all videos and animations for problem 1 and 2.
   - Modified and improved visualization code provided by the instructor.
   - Writeup for section 1 and 2 in the document.

References

[1] Stilman, Mike, *Task constrained motion planning in robot joint space.*, Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on. IEEE, 2007

[2] Kunz, Tobias, and Mike Stilman, *Manipulation planning with soft task constraints.*, Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on. IEEE, 2012