AIMS CDT Project Report: Motion Planning To Smoothly Intercept Moving Objects

Mark N Finean\textsuperscript{1}, Ioannis Havoutis\textsuperscript{2}

\textsuperscript{1}AIMS CDT, University of Oxford
\textsuperscript{2}Department of Engineering Science, University of Oxford

Abstract—Kinematic planning techniques are well researched and have been shown to be very effective in scenarios with static objects. However, these approaches fail when interacting with dynamic environments where objects are moving. Among the key challenges faced in picking up moving objects are that planning needs to account for firstly whether the solution will intercept the object at an appropriate time, and secondly obey the constraint that we typically don’t want a large deceleration, in order to avoid breakages and spillages.

In this research we build on the search-based kinodynamic motion planning algorithm presented by Menon et al. which generates a time-parameterised trajectory for whole-body motion in order to intercept an object at the earliest feasible point in it’s trajectory. Their research focussed only on planning in two spatial dimensions, working well in a conveyor belt environment at a fixed height. In this research we propose a fast, on-the-fly heuristic to counter the high dimensionality of the problem and expand the search space to plan in a 3D environment, greatly increasing the capabilities of the robot. We present the progress achieved so far and describe our plans for continuing this research.

We verify our implementation in simulation on a 6-DOF environment, controlling the base rotation and 5-DOF of the arm of Toyota’s Human Support Robot (HSR) \cite{menon2018}, to intercept a Cartesian goal coordinate at the earliest time.

I. INTRODUCTION

Industries and businesses continually focus on automating parts of their business pipeline to improve efficiency, reduce timescales and ultimately cut costs. In recent years, automation of warehouses has been a key focus area and is still at the forefront of robotics research. Pick-and-place tasks are often central to warehouse automation, with e-commerce giants such as Amazon and Ocado investing heavily in using robots to pack and move large bins of products \cite{menon2018} \cite{havoutis2019}. Crucially however, in these factories the objects are typically at rest. We believe that by enabling more dynamic systems, we will unlock further efficiencies in solving these tasks.

Take a moment to picture a fast-paced and efficient office with numerous workers all at work. You can see people rushing between tasks, passing papers between them as they walk past each other. You see a worker race past their desk and pick up some documents on the side without breaking stride. Another worker sweeps a coffee mug off a table since they’re already going past the kitchen. The picture we’re painting here is a dynamic one and it is this fluidity and efficiency in combining tasks that still distinguishes robotic capabilities from that of humans.

State-of-the-art manipulators still appear very ‘robotic’ in their movements and typically execute separate planning stages for getting close to an object, picking up the object, and then moving again. By integrating the planning for multiple tasks, we believe that robots could achieve a more human-like grace in their movements, as well as improve their efficiency.

An increasingly prevalent field is that of ‘Human Support Robots’. These are robots that can assist and interact with us on a daily basis. To improve the relationship and experience that humans have with robots, their movements need to appear more natural.

To address these issues we explore how search-based planning can be used to provide solutions for interacting with moving objects. We build on the work of Menon et al. \cite{menon2018}, which focussed on planning in 2D, and expand the search to 3D, greatly enhancing the capabilities of the robot.

Naturally, by increasing the dimensionality of the problem, the search-space increases exponentially. We minimise planning time by addressing the importance of a good heuristic to guide the search more effectively and thus propose a fast-to-compute heuristic that is generated on-the-fly.

We frame the planning problem as a graph and use velocity motion primitives to construct the edges between state vertices. We use the ARA* algorithm \cite{moreno2019}, an anytime variant of heuristic search, to search the graph for an allotted time period and generate a minimal time-cost trajectory for the end effector to reach a goal state while avoiding obstacles and being subject to the velocity and acceleration constraints of the robot.

II. RELATED WORK

The majority of path planning algorithms only consider the kinematics of the problem \cite{kaelbling1988} such as directions that a manipulator can move in and whether there are obstacles to be avoided. However, motion planning for a robot manipulator is an inherently dynamic problem. The manipulator will be subject to velocity and torque constraints and, if interacting with moving objects, the motion is further constrained by the dynamics of the object as well. Accounting for these dynamic constraints in the motion planning is known as ‘kinodynamic’ planning.

Motion planning problems have been approached in numerous ways in the literature. Sampling-based planning
algorithms have been popular in recent years with significant research into Probabilistic RoadMaps (PRMs) [5] and variants of Rapidly-exploring Random Trees (RRTs) in particular [10] [8] [6]. These methods are quite robust to the curse of dimensionality and can achieve very short planning times. However, they typically focus purely on finding a feasible path quickly but at the expense of cost minimisation, resulting in long trajectories and ‘jerky motions’ [9].

Our research is interested in finding the time optimal solution to the kinodynamic planning problem in order to successfully interact with moving objects. To achieve this we have built on the work of Menon et al. They [4] present a kinodynamic motion planning algorithm which generates time-parameterised trajectories for the entire pickup motion of a moving object. This approach uses a search-based planning, which yields “more consistent solutions between planning episodes with similar start and goal configurations” [9]. ARA* is used to search the graph constructed.

However, they plan only in two spatial dimensions and fix the height at which the robotic manipulator can work at. This works well for very handpicked scenarios, such a robot operating on a conveyor belt which is always at a fixed height. However, for the purpose of a human support robot, we want the robot to be able to interact with objects all over the home, ranging from bottles on tables and books on shelves, to tidying up objects on the floor. We thus expanded on their work and extended the search algorithm to three dimensions.

To overcome the obstacle of high dimensionality, we developed an accurate, fast-to-compute heuristic to guide the search and reduce planning time.

## III. PLANNING PIPELINE

### A. Graph Construction

We firstly represent our planning problem as a graph, \( G \), discretising the kinodynamic configuration space into the set of states \( S \) and the possible connections \( E \). We thus construct the graph to search \( G = (S, E) \). For the end effector of the HSR to reach any 3D coordinate in the local vicinity we required 6-DOF (1-DOF in the base rotation, and the other five in the arm).

We discretised velocity and angle states into 100 equal divisions between their minimum and maximum values. The permitted edges \( E \) consisted of a fixed set of motion primitives between states, where the velocity of one joint can be changed at a time to any of the discrete velocities allowed for that joint as long as torque and velocity constraints are obeyed. We can make this assumption because the maximum acceleration constraint on all joints on the HSR is such that as long at the discrete time steps are greater than 0.25s, the maximum velocity can be reached after one time step for any joint.

Formally, we can express an individual state \( s \) as the tuple,
\[
s = (\theta_1, \ldots, \theta_6, \dot{\theta}_1, \ldots, \dot{\theta}_6, t),
\]
(1)

The fixed set of motion primitives \( E \) then take the form,
\[
e = (\Delta \dot{\theta}_1, \ldots, \Delta \dot{\theta}_6, \Delta T),
\]
(2)

where \( e \in E \).

We want to solve for the trajectory that corresponds to the minimum time cost; we attribute a cost of one time step per edge, corresponding in our case to 0.3s. Minimising the cost thus corresponds to minimising the time taken to reach the goal state.

We currently use inverse kinematics to determine the joint angles corresponding to the goal state and provide this as input for the planner. In future research we hope to integrate kinematics from that. For now, this is provided.

### B. Searching

We search \( G \) using the ARA* algorithm, an anytime variant of heuristic search. At each time-step, the planner traverses each of the motion primitives to find the successive states. We calculate the states by integrating the motion over one time-step assuming constant acceleration between successive states. For each of the successive states, a heuristic value is calculated to guide the search.

### C. Heuristic

There are broadly two categories of motion planner - those that are ‘complete’ and those that are ‘heuristic’ [11]. Complete motion planners are guaranteed to find a solution if one is there, and if not then it proves there isn’t one. In contrast, a heuristic motion planner, as the name suggests, uses a heuristic value to provide a lower estimate of graph edge costs in order to guide the search algorithm. A heuristic search algorithm will typically find a solution much faster than a complete one but may fail to find a solution even if there is one. The nature of planning the movements of a robot manipulator in a dynamic environment is a high-dimensional problem and naturally time constrained.

Besides the heuristic guiding the algorithm to find a fast solution, ARA* has the advantage that it will progressively tighten the sub-optimality bound over time and try to improve on a solution for an allotted amount of time [12]. This enables us to customise the algorithm to individual tasks, inputting how we would like to prioritise planning time and the length of the trajectory. Given enough time, the algorithm will find a provably optimal solution, as long as the heuristic is admissible (underestimating or equal to the true cost).

The choice of heuristic is of paramount importance. In the limit that the heuristic goes to zero for all states, ARA* becomes Dijkstra’s algorithm and the exponential combinatorics in high dimensions leads to unrealistic planning times. At the other extreme, the heuristic must be strictly less than equal to the true cost in order for algorithm to converge on a solution. The aim is thus to use an accurate heuristic.

In our research, we are interested in finding the fastest/lowest cost solution in order to intercept an object at the earliest point in it’s trajectory. We thus make approximations for the minimal cost in order to construct a fast-to-compute algorithm on the fly, shown in Algorithm 1.
The accelerations possible on the HSR are such that we can always reach the maximum velocity in less than 0.3s. By constructing our step size to be at least 0.3s, we can work in velocity space and select velocities as motion primitives. This assumption also enables the construction of our heuristic algorithm.

Algorithm 1

1: \(\text{timeStep} \leftarrow 0.3\)s
2: \(\text{function GETHEURISTIC}(S1, S2)\)
3: \(\text{fromState} \leftarrow S1\)
4: \(\text{toState} \leftarrow S2\)
5: \(\text{for } i \text{ in range(numOfJoints)} \text{ do}\)
6: \(x_i \leftarrow \text{toState.angles}[i]\)
7: \(x_f \leftarrow \text{fromState.angles}[i]\)
8: \(v_i \leftarrow \text{fromState.velocity}[i]\)
9: \(v_f \leftarrow \text{fromState.velocity}[i]\)
10: \(v_{\text{max}} \leftarrow \text{max speed for joint } i\)
11: \(d_1 = d - \frac{1}{2} \times \text{timeStep} \times (v_i + v_f)\)
12: \(d_2 = x_1 + (\text{timeStep} \times v_{\text{max}})\)
13: \(d \leftarrow x_f - x_i\)
14: \(\text{if } d == 0 \text{ then}\)
15: \(\text{if } v_i == v_f \text{ then}\)
16: \(\text{heur}[i] = 1\)
17: \(\text{else}\)
18: \(\text{heur}[i] = 0\)
19: \(\text{else}\)
20: \(\text{if } \text{sign}(d) \times (d - x_1) < 0 \text{ then}\)
21: \(\text{heur}[i] = 2\)
22: \(\text{else}\)
23: \(\text{if } \text{sign}(d) \times (d - x_2) < 0 \text{ then}\)
24: \(\text{heur}[i] = 2\)
25: \(\text{else}\)
26: \(m = \text{abs}(\text{ceil}(\frac{d - x_1 - x_2}{v_{\text{max}}}))\)
27: \(\text{heur}[i] = 2 + m\)
28: \(\text{return } \text{sum(heur)}\)

D. Forward Kinematics

Built into the planner is also a method of collision avoidance. We discretised the environment into 3D voxels, labelling voxels whether they were safe or obstacles, enabling us to calculate whether successive states are safe. We also use this 3D environment to inform us whether the Cartesian goal state has been reached. We do this by using forward kinematics on the joint angle state to calculate the Cartesian position of the end effector. This is done by using OpenRAVE [13] to construct a model of the robot as shown in Figure 1. We then set the joint angles for each successive state and return the position of the end effector.

Fig. 1: The HSR robot model generated by OpenRAVE using the robot URDF file. By setting the joint angle values of the model, we used OpenRAVE to conduct the forward kinematics on successive states in order to find the effector position in Cartesian space. This enabled obstacle avoidance checking as well as method of determining when the goal position had been reached.

IV. Results

The preliminary results of this work has shown the use of ARA* in conjunction with our on-the-fly heuristic to be effective in generating a short, time-optimal trajectory. Executing the planner on 10 different goal positions we found the planning time to range between 4.7s and 45s, suggesting that further work is required to reduce planning time.

We simulated our planner solutions on the HSR simulation (Figure 2) and we found the end-effector to successfully reach the goal position while avoiding obstacles.

V. Conclusion

In this paper we have demonstrated the use of search-based motion planning, via the use of ARA*, to generate an accurate time-optimal trajectory for a robot manipulator in simulation. The planner incorporates collision avoidance, generates a smooth trajectory and operates effectively in 3D Cartesian space. We proposed and implemented our own heuristic to deal with the high dimensionality of the problem. Preliminary results show the success of the planner and we now intend to focus on reducing the planning time and implement this in a dynamic environment as part of a full manipulation pipeline in order to grasp moving objects.
Fig. 2: The solutions output from our motion planner were tested in simulation on the HSR robot. As shown in Figure 2a, the robot was initialised to a certain state and then a goal location was provided. Figure 2b illustrates the accuracy of the resultant trajectory generated.

VI. FUTURE WORK

We created a lean simulation environment, shown in Figure 3, for testing the planner on static and moving objects. We intend to integrate the simulation environment with the motion planner and a Kalman filter to provide goal positions for intercepting a moving object. As previously discussed, achieving minimal planning time is crucial in enabling the planner to find solutions in an acceptable time-frame to achieve its task. At present the planning time required for this planner is too large to be used on fast moving objects and so we aim to make improvements to the heuristic, develop a leaner code base to reduce the planning time, and optimize the motion primitives.

Fig. 3: A simple simulation environment in Gazebo in which we intend to test the planner on objects rather than manual goal positions.
REFERENCES


