Autonomous Forklift Implementation

Gudjon Einar Magnusson
gmagnusson@fc-md.umd.edu

***Abstract*— In this paper I present an implementation of an autonomous forklift uses a combination of task and motion planning to navigate in a warehouse-like environment and picks and places cargo pallets to satisfy a goal. The forklift uses propositional logic to plan high level task plans and then uses RRT to navigate as it performs each step of the plan.**

I. Introduction

Task planning and motion planning are both mature fields that offer a verity of advanced algorithms to solve their respective problems. These domains are generally studded in isolation by different people and surprisingly little consideration is given to the interaction between the two. It seams clear that any autonomous robot requires cooperation between task and motion planning.

Task planners can find a long sequence of actions to carry out a task under complex logical constraints. These planner abstract away the properties of the physical world they operate in. Motion planners can find a optimal or near optimal path to move a physical object through space while avoiding collisions and respecting dynamic constraints. They generally operate over the time scale of a single task but do not consider the big picture.

In this paper I want to explicitly address how the two domains can be brought together in a single implementation. For demonstration I use an autonomous forklift operating in a warehouse-like environment. I believe this is a good example because it is fairly simple and intuitively shows the need for the combination of task and motion planning. Warehouse automation has also become one of the largest market for autonomous robotics

In section I I describe the details of the vehicle being used. I section II-C.0.c I describe the planning domain and the methods used for high level task planning. In section III-B I describe the motion planning methods used to navigate.

II. Vehicle Description

A. Vehicle Dynamics

The vehicle is a standard forklift as one would expect to find in a warehouse. The forklift is driven by its two front wheels a turns using two back wheel. For this work I don't consider the full blown reverse Ackerman steering model, but instead use a simplified car model. The simplified model assumes instantaneous change in steering and velocity. The state of the vehicle can therefore be described by a 3 dimensional configuration space (x, y, θ), and the vehicle control is a 2 dimensional input vector (v, s), where v is the target velocity and s is the steering angle. The vehicle dynamics are described by the following equations:

$$\dot{x} = vcos\theta \tag{1}$$
$$\dot{y} = vsin\theta \tag{2}$$
$$\dot{\theta} = \frac{v}{L}\tan s \tag{3}$$

In addition to driving, the forklift is able to lift and lower its forks. For the purpose of navigation the state of the forks is not considered. The forks are considered to part of the forklifts footprint and are not allowed to overlap obsolesces in the environment. There are some scenarios were lifting the forks up to pass over obstacles would enable some solutions that would otherwise be impossible, but I concluded that this was not worth the effort. The one exception to this is when the forklift actually picks up a pallet. For that action the forks need to overlap the pallet as they are threaded through the holes on the pallet. This process is not controlled by the motion planner used for navigation, but instead uses a closed-loop controller that monitors the forklifts alignment with the pallet as it slowly moves forward.

B. Simulation

The forklift is implemented using the robot operating system (ROS) and Gazebo for simulation. For this work the planner uses perfect information read directly from the simulator, the problem of localization and sensing is not considered. It is assumed that the forklift has knowledge of its environment and its own position. Adding simultaneous localization and mapping (SLAM) for localization and vision for pallet identification has been considered for future work but will not be described here.

C. Command Set

The forklift can be controlled using 3 high level commands. Each command is implemented with a closed loop controller.

a) Move(Path): Given a path (a list of positions) the forklift will follow the path as well as it can. If for any reason the forklift ends up to far off the path the action is canceled and return failure.

b) Pickup(TargetPallet): Given a target pallet the forklift will move carefully to thread the forks through the holes on the pallet. It is assumed that the forklift is already close to alignment with the pallet before the command is executed. If not it is likely to fail.

c) PutDown(TargetPose): Given a target pose the forklift will carefully move to align with the pose such that it can lower a pallet it is holding down on the target pose.

III. Task Planner

Task planning for the forklift is performed using a STRIPS-like planner. The discrete state space is described using propositional logic and actions are described as a set of preconditions and a set of effects. The state space can be navigated by applying the effects of an applicable action to a state to get the next state. An action is applicable in a state if all of its preconditions are true in that state.

A. Planning Domain

Objects:

- $Robots = \{forklift\}$
- $Pallets = \{pallet_1, pallet_2\}$
- $Stacks = \{stack_1, stack_2, stack_3\}$

Predicates:

- $On(p, s)$, pallet p is on stack s
- $Holding(r, p)$, Robot r is holding pallet p
- $Empty(s)$, stack s is empty
- $Aligned(r, s)$, robot r is aligned to stack s

Actions:

- $Align(r, s)$, move robot r so it is aligned to stack s

$$\begin{aligned} pre:&\ \neg Aligned(r,s)\\ eff:&\ Aligned(r,s)\end{aligned}$$

- $Take(r, p, s)$, robot r picks up pallet p of stack s

$$\begin{aligned} pre:&\ Aligned(r,s),\ Holding(r,None),\ On(p,s)\\ eff:&\ Holding(r,p),\ \neg On(p,s),\ Empty(s)\end{aligned}$$

- $Put(r, p, s)$, robot r puts pallet p on stack s

$$\begin{aligned} pre:&\ Aligned(r,s),\ Holding(r,p),\ Empty(s)\\ eff:&\ Holding(r,None),\ On(p,s),\ \neg Empty(s)\end{aligned}$$

B. Forward Search

To construct a plan that reaches a given goal, the forklift uses breadth-first-search(BFS) to search a state graph from the start state until a state that satisfies the goal is found. BFS guaranties that the shortest solution is found, if one exists, but its not very efficient. The branching factor of this planning domain is quite high, especially as the number of pallets and stacks increases. To find a plan with 8 actions, involving 2 pallets and 3 stacks requires evaluating about 2800 nodes.

IV. Motion Planner

To perform actions the forklift needs to navigate from place to place. The motion planner plans a path for the forklift to follow from one pose to another, while respecting the kinematic constraints of the forklifts motion. This is implemented using the RRT algorithm.

To be able to pick and place cargo pallets the forklift needs to arrive at a precise location and orientation so that the forks align with the holes on the pallet. Precisely connecting two poses in configuration space requires solving the two-point boundary value problem (two-point BVP). For a simplified car model like the forklift, the two-point BVP can be solved using Reeds-Shepp curves. Reeds-Shepp curves provide an optimal path for a car-like vehicle with a fixed turning radius and can move forward and backwards, as apposed to the Dubins curves that assume only forward motion.

When planning a path the planner must account for obstacles in the workspace. Pallets are generally considered to be obstacles that must be avoided but unlike static obstacles like walls, their status can change depending on the state of the task. When the forklift picks up a pallet it is removed from the list of colliders in the workspace, when the pallet is put down the collider is added back.

V. Task example

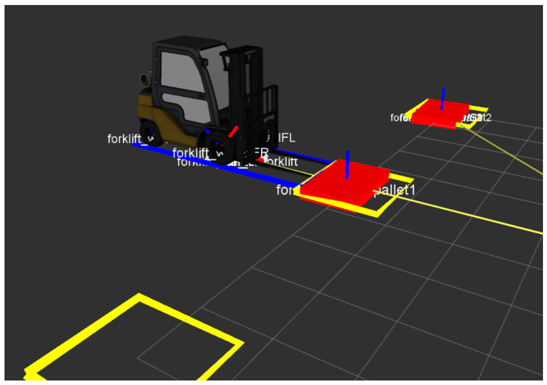

Fig. 1. The forklift working on a task involving 2 pallets and 3 stacks. Visualized in Rviz

A good example task is a two pallet swap using one extra stack for temporary storage. This scenario is shown in figure 1 and is specified using the following start state and goal.

Start State:

- $Holding(forklift, None)$
- $On(pallet1, stack1)$
- $On(pallet2, stack2)$
- $Empty(stack3)$

Goal:

- $On(pallet1, stack2)$
- $On(pallet2, stack1)$

For this scenario the shortest plan that satisfies the goal involves 12 actions:

- $Align(forklift, stack1)$
- $Take(forklift, pallet1, stack1)$
- $Align(forklift, stack3)$
- $Put(forklift, pallet1, stack3)$
- $Align(forklift, stack2)$
- $Take(forklift, pallet2, stack2)$
- $Align(forklift, stack1)$
- $Put(forklift, pallet2, stack1)$
- $Align(forklift, stack3)$
- $Take(forklift, pallet1, stack3)$
- $Align(forklift, stack2)$
- $Put(forklift, pallet1, stack2)$

To execute the plan returned by the task planner, each of the actions must be translated to a forklift command and executed one by one. For each move action the motion planner is called to plan a path.

VI. Conclusion

In this paper I presented an autonomous forklift that uses a combination of task and motion planning. Unfortunately it does not address the problem of pickling arbitrary stack location to solve problems like initially wanted to do.