Motion Planning for a Drive-by Pick-And-Place Application

Robert Shi

Oriel College

Engineering Science Department

University of Oxford

May, 2020

Oxford, United Kingdom

Motion Planning for a Drive-by Pick-And-Place Application

Robert Shi

Oriel College

Engineering Science Department

University of Oxford

May, 2020

Oxford, United Kingdom

1 Acknowledgements

I would like to express my deep gratitude to my project supervisor, Dr. Ioannis Havoutis, for his guidance and patience throughout this project. I am truly grateful for the opportunity to learn about robotics and gain hands-on experience as an undergraduate student.

I would also like to express my appreciation to Dr. Wolfgang Merkt for setting up the drive-by problem in EXOTica and for his valuable advice and assistance.

Finally, I would like to thank Mark Finean and Charlie Street for their time and support in helping me understand and operate the Toyota Human Support Robot.

2 Abstract

Service robots for home care applications require the ability to perform pick-and-place actions for object manipulation. It is common to split the pick-and-place task into navigation and manipulation while the robot is stationary. This project builds on prior work on loco-manipulation planning and demonstrates a continuous pick-and-place motion. This paper introduces a drive-by pick-and-place motion planning method in which a robot follows a predefined base trajectory and performs a grasping action without stopping the base motion. In particular, the drive-by pick-and-place method is implemented on a Toyota Human Support Robot to achieve time-efficient, collision-free, whole-body trajectories in a static environment. Failure modes are documented and analyzed and an outline of future work is provided.

Contents

3 Introduction

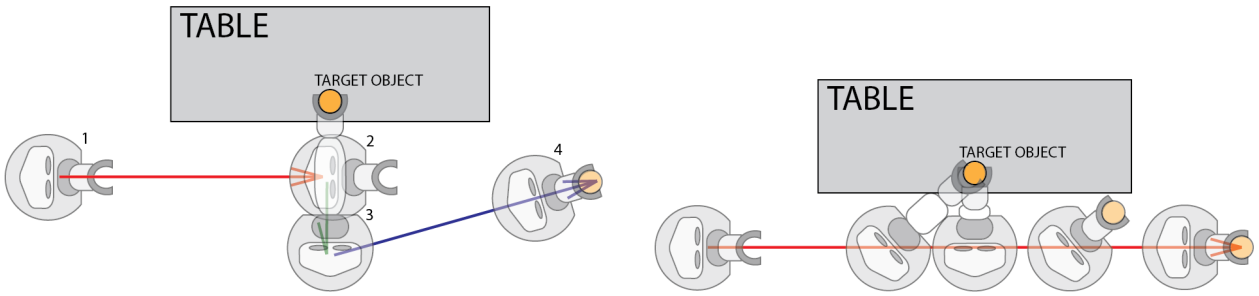
3.1 Purpose

Robocup@Home is an international robotics competition that aims to pose a standard problem to further development of robotic technologies in the area of "service and assistance with high relevance for future personal domestic applications"[1]. Specifically, the Domestic Standard Platform League (DSPL) of Robocup@Home aims to "assist humans in a domestic environment, paying special attention to elderly people and people suffering from illness or disability."[1]. Focus areas of the competition include "Human-Robot-Interaction and Cooperation" as well as "object manipulation"[1]. A common activity for an autonomous robot to perform is object retrieval or placement, more commonly known as a pick-and-place problem in robotics. This fourth-year project centers on implementing a drive-by pick-and-place algorithm on a Toyota Human Support Robot (HSR), which is the designated robot for the Robocup@Home DSPL.

The HSR software includes a function that commands the robot to position its end-effector to grasp an object and a function that commands the robot to drive to a designated point in the world. The combination of these two functions allows the HSR to complete a pick-and-place task. However, the actions are discontinuous and require the HSR to stop and grasp the target object.fig. 1(a) The drive-by pick-and-place algorithm outlined in this paper aims to combine the tasks of grasping the object and driving to a target placing location to identify a time-efficient, collision-free trajectory.fig. 1(b)

3.2 Problem Description

The goal of this project is for the HSR to perform a drive-by pick-and-place action. This requires the HSR to drive by a table, following a predefined base trajectory, and pick up a cylindrical object on top of the table without stopping the base motion. The HSR then brings the object to the target placing location and places it. It is assumed that the base trajectory is given prior to planning and the location of obstacles is known .fig. 1(b)



(a) Pick-and-place using HSR functions

(b) Proposed drive-by pick-and-place

Figure 1: Depiction of pick-and-place methods

3.3 Definition and Notations

3.3.1 Joint Types

Robot joints fall into a few distinct classes. The HSR has 3 main types of joints: continuous, prismatic, and revolute. The continuous joints include the wheels of the base, which have one rotational degree of freedom and can rotate without a limit on the angle. The HSR has prismatic joints that lift the arm and the torso. A prismatic joint has one linear degree of freedom and allows for extension and retraction. The majority of the HSR's joints are revolute joints. Revolute joints allow for one degree of rotational freedom, but have limits on allowable joint angles.[2] fig. 2

3.3.2 Task and Configuration Space

Path planning for mobile robots occurs in both task space ($\mathcal{T}$) and configuration space ($\mathcal{C}$). Task space represents the set of all possible translations and rotations for the robot and its components in Euclidean space. For real robots, $\mathcal{T} \subset SE(2)$ or $\mathcal{T} \subset SE(3)$, depending on whether or not the robot's motion is confined to a plane. SE(n) denotes the Special Euclidean group, which includes any combination of translations and rotations in dimension n. For example, the HSR base is confined to the xy plane and so the position can be represented by (x, y, θ). The HSR gripper end-effector can move in three dimensions and has a task space representation of form $(x, y, z, \theta_{roll}, \theta_{pitch}, \theta_{yaw})$.

The robot configuration $\mathbf{q}$ is represented by joint coordinates, which specify the degree or length value for each robot joint. As a result, $\mathbf{q}$ has the same number of elements as the number of degrees of freedom (N) of the robot. The configuration space, $\mathcal{C}$, represents the set of all possible $\mathbf{q}$. $\mathbf{q} \in \mathcal{C} \subseteq \mathbb{R}^N$. $\mathcal{C}$ is separated into $\mathcal{C}_{free}$ and $\mathcal{C}_{obs}$. $\mathcal{C}_{obs}$ represents the set of all robot configurations

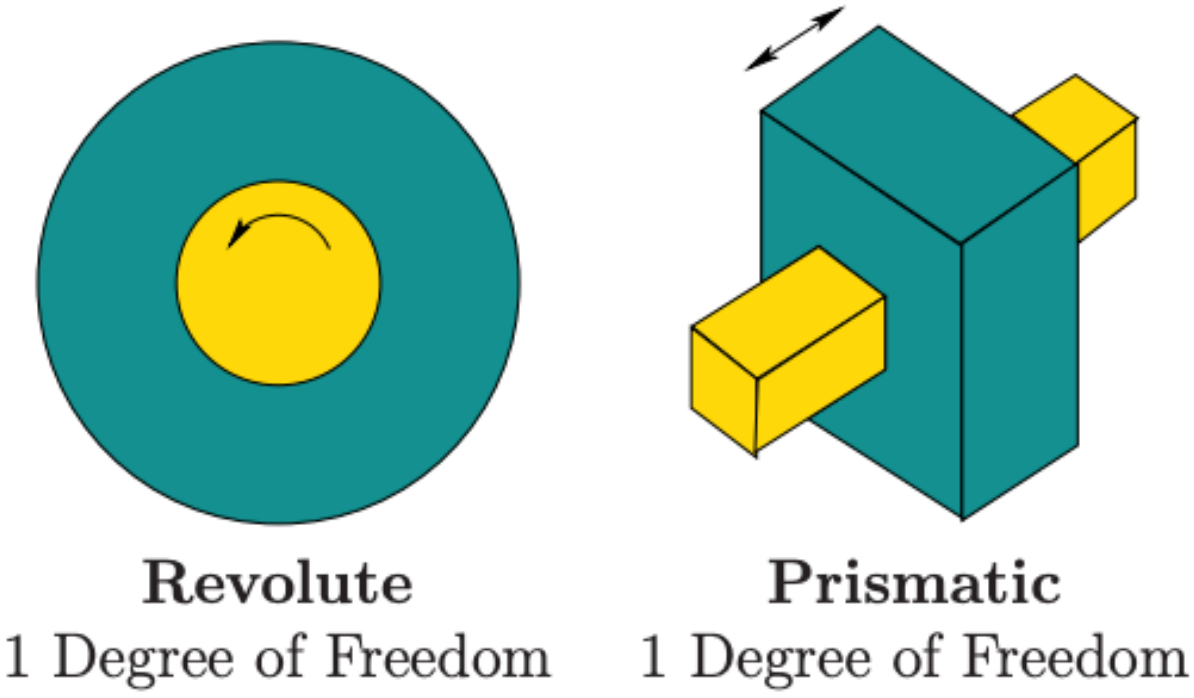


Figure 2: Types of joints. A continuous joint acts as a revolute joint without angle limits. From [2], pg. 105

that result in collision with an obstacle in task space and $\mathcal{C}_{free}$ represents the set of all robot configurations that do not result in collision. Thus, $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$.[2]

3.4 Robot Description

The HSR is one of the "Toyota Partner Robots" which aims to "provide life support and to assist with independent living in the home for handicapped people"[3]. The HSR is subdivided into 4 segments: the base, the torso, the arm, and the head.

3.4.1 Base

The HSR has an omnidirectional wheeled base with two rear drive wheels and two passive casters for locomotion as depicted in fig. 4. A continuous base-roll-joint [1]in the HSR omnibase segment allows the body of the HSR to rotate freely of the base. The base is cylindrical and houses a bumper sensor and laser range sensor to aid in obstacle detection. A magnetic sensor allows users

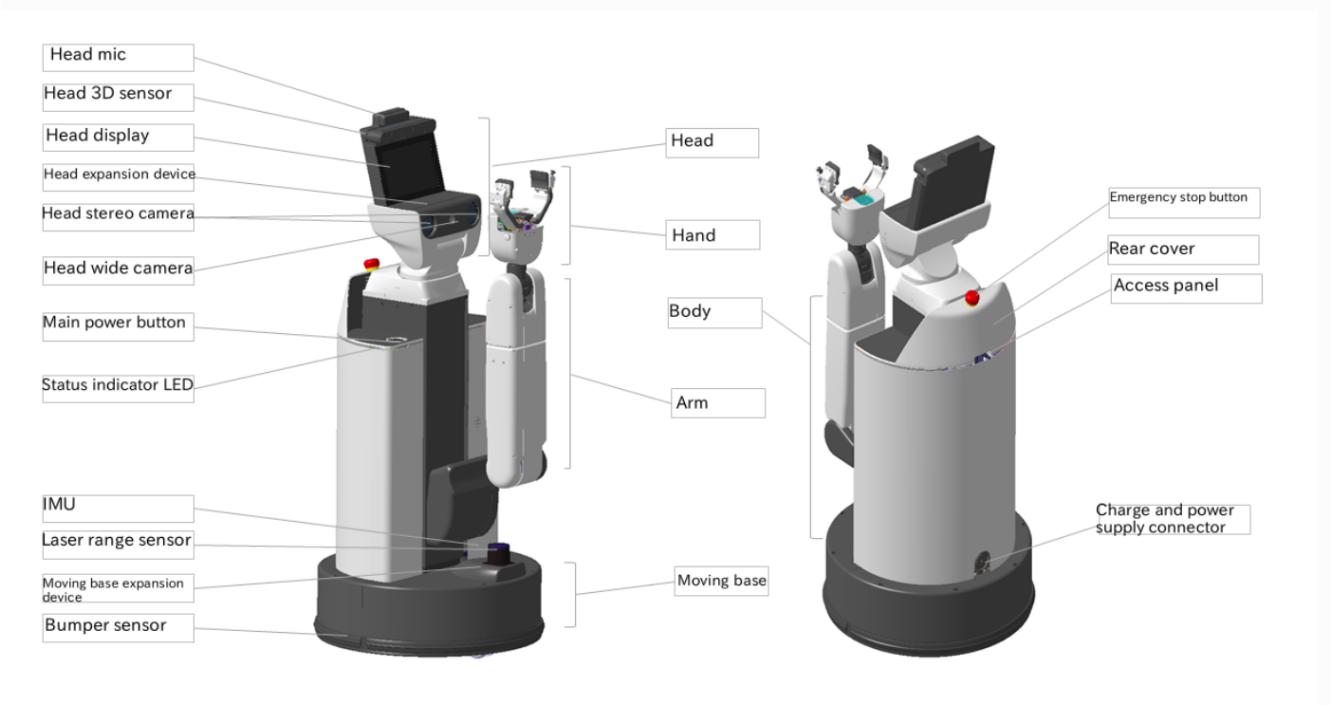


Figure 3: Overview of HSR Segments and Components.[3]

to mark world boundaries using strips of magnetic tape.

The HSR omnibase implements a dual-wheel caster drive mechanism first described in [4]. The dual-wheel caster mechanism works similarly to a classical differential drive system. A differential steered robot has two separately driven wheels on either side of the robot. The robot can travel in a straight line when both wheels are driven at the same speed in the same direction. The robot turns when the wheels are driven at different speeds. The kinematics of a differential drive is,

$$\begin{bmatrix} \dot{x}_o \\ \dot{y}_o \\ \dot{\theta}_o \end{bmatrix} = \begin{bmatrix} r/2 & r/2 \\ 0 & 0 \\ r/W & -r/W \end{bmatrix} \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} \tag{1}$$

where r is the wheel radius, $\omega_R and \omega_L$ are the angular velocities of the right and left wheel, and W is the distance between the wheels. Velocities are defined with respect to the midpoint between the wheels (O_o), which coincides with the center of the robot base in a differential drive robot. This allows a differential drive robot to turn in place. The velocity in the y-direction is always zero, so a differential drive vehicle cannot move sideways. Because the local degrees of freedom of movement

[1]The coordinated for the HSR omnibase are defined such that the front of the torso points in the z-direction and the x-direction is perpendicular to the plane of the base.

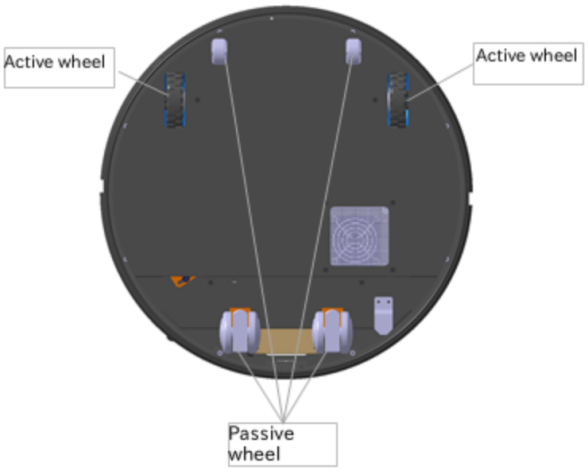


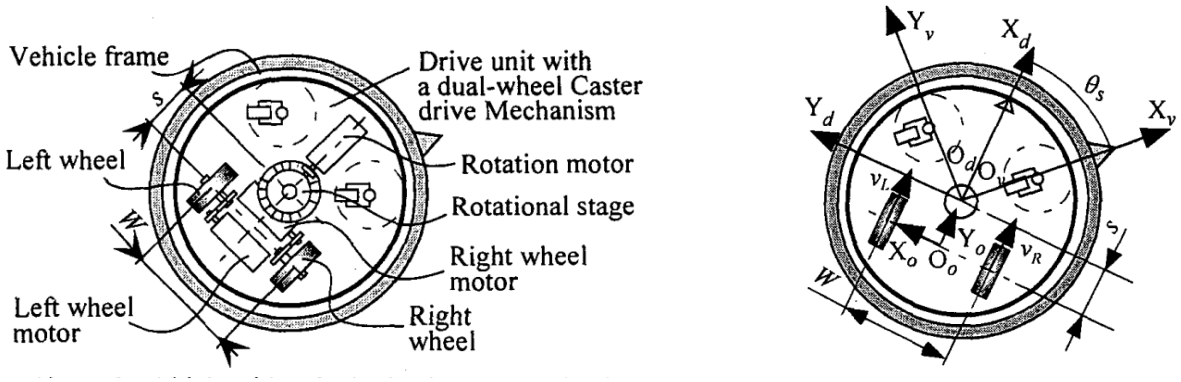
Figure 4: Omnibase of HSR [3]

for a differential drive robot is less than the global degrees of freedom, differential drive robots are non-holonomic. Holonomicity is important because motion control for a holonomic robot is easier than for a non-holonomic robot.[5] For example, instead of having to parallel-park a car, which is a non-holonomic vehicle, a holonomic vehicle can simply drive sideways into a parallel parking space.

The dual-wheel caster mechanism removes the holonomic constraint in the y-direction by offsetting the midpoint between the driven wheels and the center of the rotational stage by a distance s as shown in fig. 5(b). The rotational stage is analogous to the body of the HSR which rests on top of the base-roll-joint. This is depicted as the base-link frame in fig. 6. This offset results in the same kinematics as differential drive with respect to O_o. However, the kinematics with respect to the center of the rotational stage becomes,

$$\begin{bmatrix} \dot{x}_d \\ \dot{y}_d \\ \dot{\theta}_d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & S \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_o \\ \dot{y}_o \\ \dot{\theta}_o \end{bmatrix} = \begin{bmatrix} r/2 & r/2 \\ rs/W & -rs/W \\ r/W & -r/W \end{bmatrix} \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} \tag{2}$$

The shift in wheel placement results in a robot that has a non-zero velocity in the y-direction when turning. The dual-wheel caster drive mechanism resolves the non-holonomic constraint of differential drive-by forgoing the ability to turn in place. The omnidirectional base allows the HSR to take a direct path to a goal point, which simplifies paths.



(a) Dual-wheel caster mechanism

(b) Coordinate system of the vehicle

Figure 5: Visualizations for dual-wheel caster mechanism from [4].

3.4.2 Body and Head

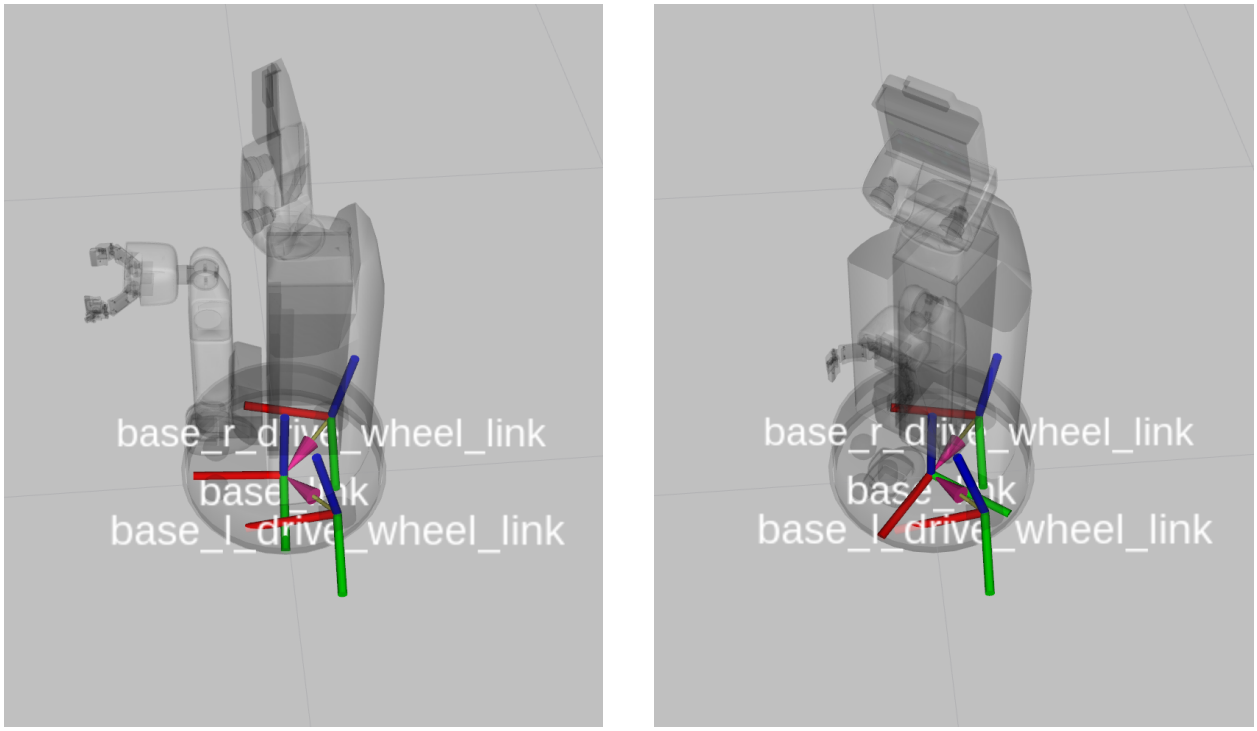
The HSR has a height range between 1.0m and 1.3m due to a telescoping body. The head contains a wide angle camera, stereoscopic camera, and an ASUS Xtion depth sensor for visual input and object recognition. In addition, there is a microphone and screen display for human input. The head can pan and tilt.

3.4.3 Arm and Hand

The arm is attached to the body by a lifting shoulder joint that vertically telescopes the head and arm. The arm, including the lift joint, has 5 degrees of freedom and a length of 0.60 meters. The end effector is a two-finger rubber gripper (hand) with a camera, force sensor, and suction pad. The gripper can open to a maximum of 13.5cm, apply a maximum of 40N of force, and open and close within 0.4 seconds.

3.4.4 Development

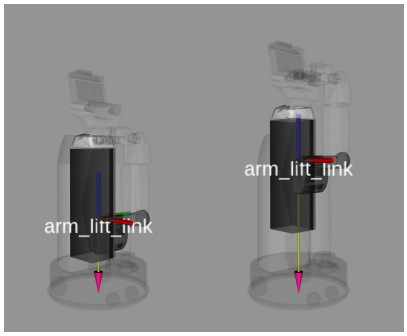
The HSR has software built on top of the Robot Operating System (ROS), which is an open-source framework used to write robot software.[6]. The HSR software includes a Python Interface and ROS interface. Toyota introduces an interactive shell based on iPython as an option for users to interface with the HSR without the steep learning curve associated with ROS. This shell interface allows for whole body motion, as well as individual subsystems by specifiying goal points, goal poses, and relative movement. Further, the HSR has a graphical web-interface for usage by



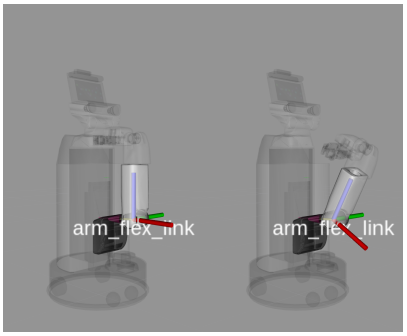
(a) (b)

Figure 6: The base roll link allows the HSR to rotate its torso without moving the omnibase. Note that the orientation of base_link changes between (a) and (b), while the drive wheels remain stationary.

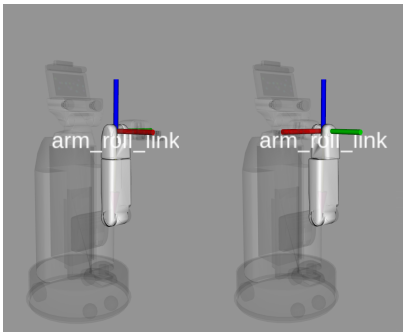
non-technical operators that allows for basic teleoperation.



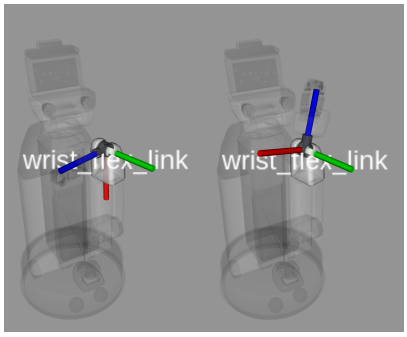
(a) arm-lift-joint



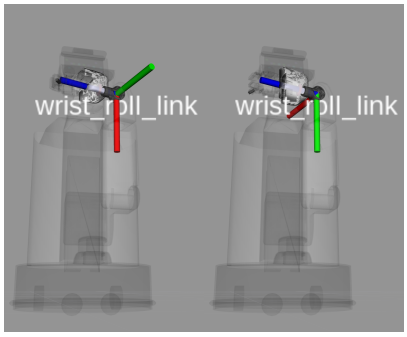
(b) arm-flex-joint



(c) arm-roll-joint



(d) wrist-flex-joint



(e) wrist-roll-joint

Figure 7: HSR arm joints

4 Planning Algorithms

4.1 Overview

To perform a drive-by pick-and-place task, the HSR must satisfy constraints such as avoiding self-collision, avoiding collision with obstacles, and maintaining a given base trajectory. This is a whole-body locomanipulation task, which is defined as actions that involve the whole body of a robot for locomotion, manipulation, or stabilization.[7]. Since the HSR is a wheeled-base robot, stability is not as large of a concern compared to legged, walking robots, for which whole-body loco-manipulation planning is mainly used. HSR joint limits prevent the robot from falling over and the maximum recommended weight of objects is 500g, much greater than the soda can be used in testing.[3] In addition to considering multiple constraints, whole-body motion planning poses the difficulty of planning motion for a large number of degrees of freedom. However, whole-body motion planning allows the robot to simultaneously drive-by and pick up an object as in fig. 1(b).

Planning the path of the HSR base and end-effector is performed by a path planner. In this project, two classes of planners are used: sampling-based planners and trajectory optimization

planners. Sampling-based approaches randomly sample the free configuration space to identify collision-free paths between a start and goal state. Optimization-based planners minimize an objective function composed of smoothly differentiable constraints to produce a collision-free and smooth trajectory. Implementations of both types of planners will be described and compared to justify their usage.

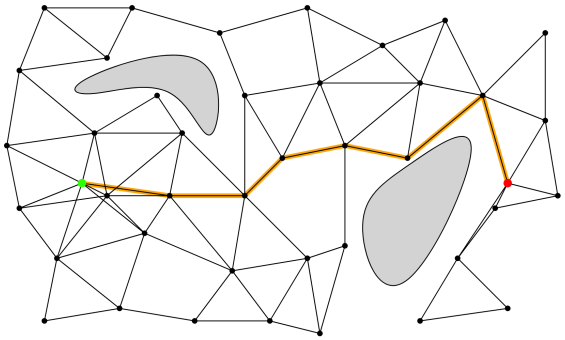
4.2 Sampling-Based Planning

Two common sampling-based planners are Probabilistic Roadmaps (PRM) and Rapidly Exploring Random Trees(RRT). Both methods are probabilistically complete, meaning that if a solution exists, it will be discovered given infinite time. A variation of the RRT algorithm is used in this project. A description of PRM is included since LaValle closely compares RRT to PRM in the original RRT paper, and thus it serves as a good starting point for sampling-based planning.[8]

4.2.1 PRM

```
(1)   N ← ∅
(2)   E ← ∅
(3)   loop
(4)      c ← a randomly chosen free
          configuration
(5)      N_c ← a set of candidate neighbors
           of c chosen from N
(6)      N ← N ∪ {c}
(7)      for all n ∈ N_c, in order of
          increasing D(c,n) do
(8)        if ¬same_connected_component(c,n)
            ∧Δ(c,n) then
(9)                E ← E ∪ {(c,n)}
(10)               update R's connected
                    components
```

(a) PRM Algorithm [9]



(b) Example of PRM [10]

Figure 8: Probabilistic Roadmap

A PRM is initialized with the starting state of the robot and a desired goal state. Robot configurations are randomly drawn from the set of free configurations. This implies that a map of the environment with obstacles and free regions is known. These random configurations are represented as nodes on a graph. Neighboring nodes with an unobstructed path and a distance below a given threshold are connected. This represents a valid path between two configurations. An issue arises here as a euclidean distance metric gives a computational speed advantage, but

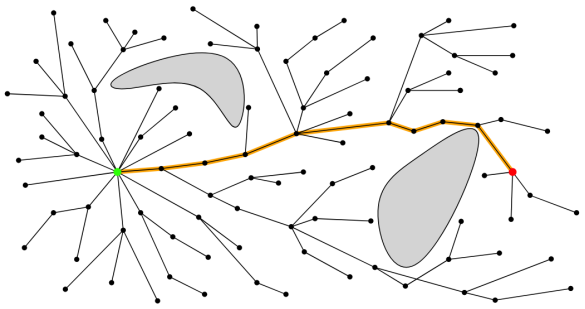
assumes holonomicity. Alternatively, [PRM paper] proposes using a local planner to calculate swept-area/volume as a distance metric, which would allow the method to extend to non-holonomic robots, but the authors note that this method is time-consuming. Given sufficient samples, the nodes would fill the entire free configuration space. A unidirectional search of the network using an algorithm like Dijkstra's shortest path would give the shortest valid path in the PRM. Given infinite random samples, the PRM method would search the entire free configuration space, meaning that it is asymptotically optimal [5]. Further, since the random sampling is not affected by the start and end states, the roadmap can be saved and queried for other start and end states, reducing computation time for subsequent planning calls. This property makes PRMs a multi-query planner, since multiple plans vith varying start and goal states can be generated from a single roadmap.

The drive-by pick-and-place method does not require many paths to be planned, so the exhaustive planning in configuration space provided by a PRM is not necessary. This motivates the need for single-query planners that can more efficiently identify a single valid path. Such an example is the RRT planner.

4.2.2 Vanilla RRT

GENERATE_RRT($x_{init}, K, \Delta t$)
1 $\mathcal{T}$.init(x_{init});
2 **for** $k = 1$ **to** K **do**
3 $x_{rand} \leftarrow$ RANDOM_STATE();
4 $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}$);
5 $u \leftarrow$ SELECT_INPUT(x_{rand}, x_{near});
6 $x_{new} \leftarrow$ NEW_STATE($x_{near}, u, \Delta t$);
7 $\mathcal{T}$.add_vertex(x_{new});
8 $\mathcal{T}$.add_edge(x_{near}, x_{new}, u);
9 Return $\mathcal{T}$

(a) RRT Algorithm [8]



(b) Example of RRT [10]

Figure 9: Rapidly-Exploring Random Trees. (a): x_{init} is the start state, K is the number of verticies, and δt is a time interval. Note, the algorithm in (a) simply explores the free configuration space.

Rapidly-Exploring Random Tree algorithm (RRT) and its variants are widely used for autonomous path planning for mobile robots. Proposed by Steven LaValle in 1998, RRT is a sampling-based planning method that is probabilistically complete and [1] naturally accommodates non-holonomic constraints. Instead of creating a cyclic graph in the case of the PRM, RRT

forms an acyclic, directed tree rooted at the starting configuration node. Like PRM, RRT is also a sampling-based planner, although when extending the graph, a randomly generated configuration is not directly added to the tree. Instead, as the name suggests, Rapidly Exploring Random Trees take a step in a random direction to explore the free configuration space. A step of a given length is taken towards the random configuration from the closest node of the tree, given that the step does not result in a collision with an obstacle. This gives a slight advantage over PRMs since only a single nearest neighbor query is required. Once a branch of the tree steps within a threshold distance of the goal, a path from the start configuration to the goal configurations is found. Since the graph is rooted, directed, and acyclical, the solution is easily found with a simple tree traversal.

Unlike PRM, which is a multi-query method, the tree for RRT is rooted at the starting node, so this single-query algorithm must be rerun for differing start and end states. RRT maintains probabilistic completeness for holonomic systems due to the random sampling of the free configuration space, given that states are uniformly sampled. LaValle explains the probabilistic completeness property in terms of a Voronoi diagram of RRT vertices. Due to random sampling, exploration is biased towards large Voronoi regions, converging to a complete, uniform (or another sampling scheme) search of the free configuration space as large Voronoi regions are iteratively reduced. [image of voronoi] RRT also more readily adapts to non-holonomic constraints compared to PRMs. This is due to the directed nature of the tree. The closest node to a random configuration can be determined by a metric that accounts for a constraint such as rotation. The path from the closest node to the new node can be found by inverse kinematics under non-holonomic constraints.

4.2.3 Bidirectional RRT Variants

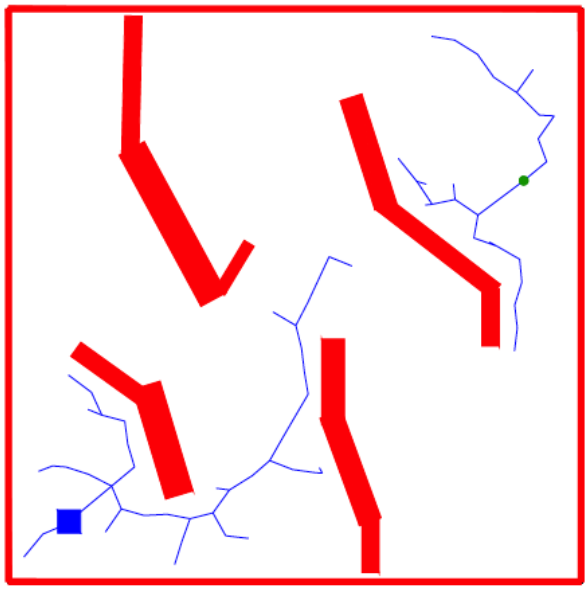
RRT has numerous variants, notably Optimal RRT (RRT*) which reorganized the tree structure as new nodes are generated. Two variants relevant to this project are RRT-Connect and Constrained BIDirectional RRT (CBiRRT2). A time-series RRT Connect algorithm is implemented in [Time Series Planning ***] which is detailed in [Section Reference]. The HSR utilizes a planner that builds upon CBiRRT2. Both RRT-Connect and CBiRRT2 are bi-directional RRT variants, meaning that trees are extended from both the start and goal nodes by swapping the start and goal trees each loop. RRT-Connect incorporates a Connect heuristic, which is a greedy function that takes the place of the Extend step in the original RRT algorithm. Instead of taking a single step towards a target configuration, q, the Connect function iteratively extends the tree towards q until either

an obstacle is encountered or the configuration is reached. LaValle and Kuffner note that q can be a random configuration or the nearest neighbor of the other tree. This allows for tuning the algorithm to prioritize either exploration or connection of the two trees.

```
CONNECT(T, q)
 1  repeat
 2      S ← EXTEND(T, q);
 3  until not (S = Advanced)
 4  Return S;
```

```
RRT_CONNECT_PLANNER(q_init, q_goal)
 1  T_a.init(q_init); T_b.init(q_goal);
 2  for k = 1 to K do
 3      q_rand ← RANDOM_CONFIG();
 4      if not (EXTEND(T_a, q_rand) = Trapped) then
 5          if (CONNECT(T_b, q_new) = Reached) then
 6              Return PATH(T_a, T_b);
 7      SWAP(T_a, T_b);
 8  Return Failure
```

(a) RRT-Connect Algorithm [11]



(b) Example of RRT-Connect [11]

Figure 10: RRT-Connect

The advantage of a bi-directional RRT over the original RRT is the expansion of the tree rooted at the goal node. As the goal-rooted tree expands, the number of possible points of connection increases, encompassing a larger portion of the free configuration space. This increases the rate of finding a feasible path. For a robot confined to motion in 2 dimensions, such as the HSR omnibase, sampling can occur in task space, by planning in x, y, θ coordinates, instead of wheel joint angles. The HSR omnibase controller contains software that converts omnibase goal positions to joint configurations using inverse kinematics. Inverse kinematics (IK) is the process of calculating joint configurations to satisfy an end-effector position in task space. Forward kinematics is the reverse process, which calculates the end-effector position in task space from a joint configuration. In general, inverse kinematics is much more difficult than forward kinematics. Since IK is the inverse of a function, the result may be no solution or infinitely many trajectories depending on the degrees of freedom of the system. For higher degrees of freedom, such as the 8 degrees of freedom HSR (3 base DOF and 5 arm DOF), it is easier to sample in configuration space, since forward kinematics is faster. In whole-body planning for the HSR, higher dimension constraint manifolds represent

the possible configurations of the robot that satisfy given constraints. The goal remains the same: to find a path from start to end states in task space that lies completely on constraint manifolds in configuration space.

4.2.4 CBiRRT

CBiRRT, the first published iteration of the algorithm, uses a method similar to the Connect-Heuristic that attempts to iteratively extend both trees towards a randomly sampled configuration. Unlike RRT-Connect, in higher dimensions, the constraint manifold may be of a lower dimension than the entire configuration space. This means rejection sampling, which is the uniform sampling the entire configuration space until a viable configuration is selected, does not work. Instead, a ConstrainedConfig step is introduced to project nodes onto the nearest constraint manifold, to ensure that configurations satisfy the given constraints. [12]

4.2.5 CBiRRT2

CBiRRT2 introduces spacial expressions called Task Space Regions (TSRs) which are used to efficiently search the configuration space, especially for end-effector manipulation with pose constraints. TSRs encode a frame in configuration space and a set of boundaries in the coordinate of the frame. TSRs are designed for sampling-based planners because they are easy to sample from and distance from a given configuration to a TSR can be quickly computed. Further, TSR's can be chained together to encapsulate multiple pose constraints on an end effector. In the CBiRRT2 algorithm, trees are grown in each iteration using one of two modes, explore and sample, determined by a set probability. The explore mode attempts to connect to a random configuration like in CBiRRT. The sample mode is a new feature that differs from CBiRRT. In sample mode, the algorithm samples from a set of TRS chains and adds a new goal node to the goal tree. One of the driving principles behind CBiRRT2, embodied by the sample mode, is the idea that a task can be accomplished in more than one way. By specifying a TSR for a goal, the CBiRRT algorithm attempts to both identify possible goal configurations that would fulfill a task, as well as solve for a path to a goal. The process of increasing the number of goal states essentially creates a pair of "seeded forests" that increase the rate of discovering a motion plan. [13]

4.2.6 HSR CBiRRT2 Variant

The HSR variant of CBiRRT2 simply varies the weighting of either arm motion or base motion. [14] Note that prioritizing arm movement results in accurate end-effector positioning, while prioritizing base movement permits the application of a larger force on an object. The HSR motion planner incorporates time-optimal path parameterization (TOPP)[14] to optimize the motion of the HSR in time under hardware constraints such as speed and torque limits. TOPP allows the HSR to reach goal poses at specified times as long as the time-from-start is greater than the optimal duration, by scaling the trajectory in time.

4.2.7 Commentary

The purpose of this project is to solve a drive-by pick-and-place problem. The HSR can already pick-and-place an object with its 'built-in' motion planner. This can be done by specifying the desired end-effector position to be reached to grasp the target object. The placement target can be specified by a second end effector target pose. The HSR can create TSRs of both target poses and two calls to the CBiRRT2 variant planner would result in a satisfactory solution to the simple pick-and-place problem. However, the drive-by component of the problem introduces a time-varying aspect that is not easily satisfied by the HSR planner.

CBiRRT2 and by extension, the HSR motion planning algorithm, address scleronomic holonomic constraints, which are defined as 'time-invariant constraints evaluated at a given configuration of the robot' [13]. In a practical application, the HSR must plan for motion in a dynamic environment with moving collision constraints such as humans in a hospital environment. [15] propose a method to plan a pick-and-place problem in time-configuration space to address such time-varying constraints.

4.3 Planning in Time-Configuration Space for Efficient Pick-and-Place in Non-Static Environments with Temporal Constraints

4.3.1 Description and Overview of the Method

[15] identifies two classes of constraints that are considered in the course of a pick-and-place problem. Type 1 constraints are differentiable and involve inequality and equality constraints, which means approaching certain desired values. These are the constraints that the drive-by

component of the problem falls into, such as end-effector positioning. Type 2 constraints are binary and include constraints such as collisions. Problems with a special case of static type 2 constraints can be solved with the sampling-based planners described in [insert section reference]. Another modification to the RRT algorithm is introduced later in [section reference] to accommodate for the time-indexed nature of dynamic environments.

[15] decomposes a time-varying pick-and-place problem into 3 segments, depending on the classes of constraints active. These segments are: reaching, grasping, and placing. The reaching and placing steps serve the purpose of bringing the robot to the target object and then to the desired object location. During the reaching and placing motions, only type 2 constraints are active, primarily collision avoidance. [15] notes that the planning for the placing segment involves accounting for the altered geometry of the end effector grasping the target object. The time-varying case differs from the solution for the simple pick-and-place problem described in [a reference to commentary section] in that the HSR must start and end each segment at a designated time for the planned robot trajectory to coincide with the trajectory of collision obstacles. During the grasping phase, the end-effector follows the trajectory of the target object until the end-effector can close and grasp the target object. In the case of the drive-by pick-and-place task, the HSR must continuously drive along a given base trajectory while keeping the gripper centered on the target soda can until the gripper can fully grasp the soda can. The time frame of the grasping phase is determined by the hardware-defined time for the gripper to close around the target object. For the HSR, the gripper closes within 0.4 seconds. The start and end configurations of the grasping phase determine the goal and start state of the reaching and placing segments. The 3 stages are concatenated to produce a complete pick-and-place trajectory.

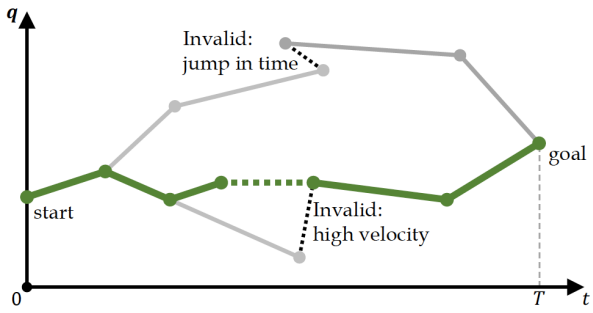
4.3.2 Time-Configuration Space RRT-Connect

Reach planning in the presence of time-varying obstacle trajectories searches for a path in time-configuration space. This space includes a time parameter in addition to the entire configuration space of the robot. The time parameter is constrained by velocity and temporal constraints. The velocity constraint ensures that joint velocity limits are not exceeded. The temporal constraint ensures causality/time-monotonicity, so the robot does not travel backward in time. Similar to the case in CBiRRT2, the sample space may be of a higher dimension than the constraint manifold, so rejection sampling does not work efficiently. This issue is further exaggerated by the introduction

Algorithm 1 Time-Configuration Space RRT-Connect

Require: $\mathbf{s}_0$, $\mathbf{s}_T$, $\mathcal{O}(t)$
Ensure: Collision-free trajectory $\mathbf{q}_{[0:T]}$
$\mathcal{T}_{forward}$.Insert($\mathbf{s}_0$), $\mathcal{T}_{reverse}$.Insert($\mathbf{s}_T$)
$\mathcal{T}_{current} = \mathcal{T}_{forward}$
$\mathcal{T}_{other} = \mathcal{T}_{reverse}$
while not Terminate **do**
    $\mathbf{s}_{rand}$ = SampleRandom()
    $\mathbf{s}_{near}, d = \mathcal{T}_{current}$.Nearest($\mathbf{s}_{\text{rand}}$)
    if $d = \infty$ **then**
        $\mathbf{s}_{rand}$ = CorrectTime($\mathbf{s}_{near}, \mathbf{s}_{rand}$)
    $\mathbf{s}_{new}$, status = Extend($\mathbf{s}_{near}, \mathbf{s}_{rand}$)
    if status **is not** Trapped **then**
        $\mathbf{s}_{near} = \mathcal{T}_{other}$.Nearest($\mathbf{s}_{new}$)
        if $\mathbf{s}_{near}$ **is not** null **then**
            status = Connect($\mathbf{s}_{near}, \mathbf{s}_{new}$)
            if status = Reached **then**
                return Path($\mathcal{T}_{forward}, \mathcal{T}_{reverse}$)
    swap($\mathcal{T}_{current}, \mathcal{T}_{other}$)
return Failure

(a) Time-Configuration RRT-Connect Algorithm [15]



(b) Example of Time-Configuration RRT-Connect[15]

Figure 11: Time-Configuration RRT-Connect

of the dimension of time. [15] introduces a CorrectTime function that works by adjusting the time parameter after a valid sample is chosen from the free configuration space. The time parameter is adjusted based on a custom distance function that only returns a real value if both time constraints are satisfied. [Reference figure 3 of time-indexed PPP*** reference].

4.3.3 Optimization-Based Planning

The grasping is solved for by a non-linear programming solver such as SNOPT and AICO. As SNOPT is a proprietary, licensed software, AICO is used for this project. AICO is a method of path planning that falls into the trajectory optimization category, unlike the sampling-based planners described so far.

4.3.4 Approximate Inference Control

Approximate Inference Control (AICO) [16] builds on Iterative Linear-Quadratic-Gaussian (iLQG) which is a method of sequential quadratic programming [17]. iLQG stems from the basic linear-quadratic gaussian control(LQG).

A linear-quadratic (LQ) problem deals with a system with dynamics that are described by linear differential equations and with a quadratic cost. An LQG control problem assumes additive, white

Algorithm 1 iLQG

Input: initial trajectory $x_{0:T}$, convergence rate α, control costs $H_{0:T}$, functions $A_t(x)$, $a_t(x)$, $B_t(x)$, $R_t(x)$, $r_t(x)$
Output: trajectory $x_{0:T}$, LQR $V_{0:T}, v_{0:T}$
repeat
  access $R_T(x_T)$, $r_T(x_T)$
  $V_T \leftarrow R_T$, $v_T \leftarrow r_T$
  for $t = T-1$ **to** 0 **do** *//bwd Ricatti recursion*
    access $A_t(x_t)$, $a_t(x_t)$, $B_t(x_t)$, $R_t(x_t)$, $r_t(x_t)$
    compute V_t and v_t using (7,8)
  end for
  for $t = 0$ **to** $T-1$ **do** *//fwd control recursion*
    compute $u_t^*(x_t)$ using (9)
    $x_{t+1} \leftarrow (1-\alpha)x_{t+1} + \alpha[A_t x_t + a_t + B_t u^*(x_t)]$
  end for
until convergence

(a) iLQG Algorithm [17]

Gaussian noise on both the state of the system as well as the measurement of the system state. An LQG controller is a control law that uses a Kalman filter for state estimation and a Linear-Quadratic Regulator (LQR) for feedback control. The LQG controller succeeds at identifying the current state and control variables at a given instant to bring the controlled process variable to the desired reference point.

The following extensions on LQG control use the principle of dynamic programming to extend LQG control to entire trajectories. Dynamic programming recursively breaks down a large optimization problem into smaller sub-problems using the Bellman equation,

$$V(x) = \max_{a \in \Gamma(x)} \{F(x,a) + \beta V(T(x,a))\} \tag{3}$$

where x is a state of the system and a is a control variable that determines the action taken to reach the next state. $V(x)$ is the optimal value of a state x. $\Gamma(x)$ denotes the set of all possible actions that can be taken at state x. $T(x,a)$ represents a state transition resulting from control variable a at state x. $F(x,a)$ is the value of taking action a at state x. β is a discount factor that decreases the value of subsequent state transitions. By finding the maximum value of the initial state x_o of a system, the optimal policy function $a(x)$ is also found, which describes the optimal action to take at a given state.

iLQG optimizes a control sequence by linearizing the system dynamics around each system state and control input in the sequence. This local linearization casts the non-linear system dynamics problem into a sequence of LQG problems, where each iteration steps through the sequence to further optimize the entire trajectory.The iLQG algorithm utilizes a forward-backward algorithm. iLQG computes a trajectory with a forward pass by applying a sequence of control inputs in an

Algorithm 2 Approximate inference control (AICO)

Input: start state x_0, control costs $H_{0:T}$, functions $A_t(x)$, $a_t(x)$, $B_t(x)$, $R_t(x)$, $r_t(x)$, convergence rate α, threshold θ
Output: trajectory $x_{0:T}$
initialize $s_0 = x_0$, $S_0^{-1} = 1e10$, $v_{0:T} = 0$, $V_{0:T}^{-1} = 0$, $r_{0:T} = 0$, $R_{0:T} = 0$, $k = 0$
repeat
  for $t = 1 : T$ **do** *// forward sweep*
    update s_t and S_t using (20)
    if $k = 0$ **then**
      $\hat{x}_t \leftarrow s_t$
    else
      $\hat{x}_t \leftarrow (1 - \alpha)\hat{x}_t + \alpha b_t$
    end if
    access $A_t(\hat{x}_t)$, $a_t(\hat{x}_t)$, $B_t(\hat{x}_t)$, $R_t(\hat{x}_t)$, $r_t(\hat{x}_t)$
    update r_t and R_t using (22)
    update v_t and V_t using (21)
    update b_t and B_t using (19)
    if $|\hat{x}_t - b_t|^2 > \theta$ **then**
      $t \leftarrow t - 1$ *// repeat this time slice*
    end if
  end for
  for $t = T - 1 : 0$ **do** *// backward sweep*
    ..same updates as above...
  end for
  $k \leftarrow k + 1$
until convergence

(b) AICO Algorithm [16]

open-loop manner. A backward pass recursively calculates the cost-to-go function, much like in the Bellman equation, except that the cost-to-go function is minimized instead of maximizing a value function. Each iteration of the backward step computes a new $a(x)$, which is applied open-loop in the subsequent forward pass. Iterating iLQG until convergence results in a locally optimal trajectory and a linear-quadratic regulator that can handle small perturbations around the selected trajectory.[17] iLQG computed updates to the entire trajectory with each pass

On the other hand, AICO takes a probabilistic inference approach by conditioning a trajectory on constraint conditions. Toussaint demonstrates that for the LQG case, the maximum likelihood trajectory approaches the solution found by cost minimization. AICO uses message-passing to propagate beliefs. In belief propagation, a selected node receives messages from parent nodes, who receive messages from their parent nodes, and so on. This propagation of information updates the selected node with information from the entire network. Then, the selected node distributes messages to its parent nodes, which redistributes information throughout the network. [18]

For trajectories, the network centers around a single chain where each node represents a time step. In AICO, beliefs are the product of received messages,

$$b(X) = \prod_i t_i(X) \tag{4}$$

where X is a random variable for which belief is expressed. t is assumed to be Gaussian for the LQG case, resulting in b also being Gaussian. The algorithm starts with approximations of t and thus also of b. Beliefs are updated by updating approximate messages according to exact messages computed by a simulator. Because of message-passing, AICO can linearize around a configuration in the trajectory instead of linearizing around the entire trajectory in the case of iLQG.[16] It is demonstrated in [16] that AICO consistently performs better than iLQG in terms of computation time to convergence.

4.3.5 Commentary

Sampling-based planners are designed to adeptly search high dimensional configuration spaces without the high computation power required for dynamic programming [Bellman, 1957]. Due to the probabilistic completeness that arises from the process of randomly sampling configurations, sampling-based planners can also better avoid the issue of becoming trapped in a local minima as in the case of gradient-descent approaches such as AICO. However, the task of keeping an end-effector centered on a target object involves multiple differentiable constraints that sampling-based planners are not able to satisfy. Also, paths planned by sampling are often not optimal due to the nature of random sampling. Optimization-based planners generally produce smooth and optimal motion. Yang, Merkt, et.al. explain that the solver gets easily trapped in local minima in static environments under collision constraints. They elect to solve grasping trajectories until a collision-free trajectory is found. [15] implemented their pick-and-place framework in the Extensible Optimization Toolset (EXOTica) framework [19]. EXOTica is "a framework of software tools designed for development and evaluation of motion synthesis algorithms within ROS" and includes an implementation of motion solvers such as AICO as a benchmark for new motion planning algorithms.[19] This project uses EXOTica to plan the grasping phase of the drive-by pick-and-place motion. Further details on the EXOTica framework are provided below.

5 Method/Approach

5.1 Overview

This project closely follows the method proposed in [15]. One significant difference is that the time-configuration planning method is designed for picking and placing a moving object. A drive-by pick-and-place scenario also has the target object moving relative to the robot, but in the drive-by case, the HSR must approximately follow a predefined trajectory which complicates the end-effector tracking of the object during the grasping phase as shown in fig. 1(b). This issue is largely due to the limited reachability of the HSR arm. To address this issue, the grasping phase is allotted an extended duration of time to encompass the HSR's approach and grasping of the object. The new, extended grasping phase plans the sub-reaching, sub-grasping, and sub-leaving steps, which are similar to the reaching, grasping, and placing steps in [15].[Insert figure to visualize the new method, with sub-divided grasping phase] reaching=¿ extended grasping phase(sub-reaching=¿ sub-grasping=¿ sub-leaving)=¿placing

This method differs from the Time-Configuration RRT-Connect in that the grasping phase is extended to incorporate a sub-reaching and a sub-leaving stage while maintaining the reaching and placing phases.

The HSR CBiRRT2 variant is used to reach the start of the extended grasping phase and leave from the end pose of the extended grasping phase to place the object. This is largely for convenience. The HSR RRT planning algorithm is built-in on the robot and the world is static for this project. The time-varying component of the problem is the position of the target object, a soda can in this instance, relative to the HSR as it drives-by, following a trajectory. The extended grasping step can be planned using AICO and activating specific constraints during the sub-grasping phase time window.

The HSR CBiRRT2 motion planning algorithm is already implemented on the HSR and TOPP allows for control over the timing of planned trajectories. This covers the reaching and placing steps of the motion plan. The key step for this project is the extended grasping phase.

5.2 Scenario Setup

To simplify the problem described in [section reference], a bare-bones world containing only an Ikea Nyboda Table and a soda can was created in Gazebo. Gazebo is an open-source robotics simulator

commonly used for ROS-based robot simulations. The objects in the simplified scenario were chosen for convenience. The Ikea Nyboda Table was chosen as it was the model of table present in the testing space for the HSR. A soda can was chosen as the target pick-and-place object since it is a household item and a representative task for a service robot. Further, the radius of the soda can is larger than items expected to be used in real-world testing, which produces a scenario with a smaller margin for error in terms of end-effector positioning. ¡insert figure¿. In this example, the HSR follows a given base trajectory that is parallel to the table and attempts to grasp the bottle.

5.3 Constraints On Grasping

The grasping method in [15] identifies valid grasping trajectories by generating new trajectories using AICO until a collision-free trajectory is found. In those cases, the grasping phases have a relatively short time frame. For instance, the HSR can close its gripper within 0.4 seconds, giving a maximum grasping duration of 0.4 seconds. In the case of the extended grasping phase for the drive-by problem, the duration for the HSR is around 15 seconds. Although computation time depends on the number of waypoints in a trajectory, the distances traversed are much larger in the case of the drive-by problem. This means that checking for collisions after computing trajectories is no longer efficient since it is highly likely that the arm or end effector would collide with the table in this test scenario. To accommodate for this issue, constraints are introduced to reduce the likelihood of colliding with the table. As mentioned above, EXOTica is used to plan this grasping phase.

5.4 EXOTica

As a benchmark framework, EXOTica introduces abstractions that separate a problem into 3 components so multiple motion solvers can be used to compare efficiency.[19] The components are a planning scene, a planning problem, and a motion solver.[ref]

The planning scene contains information about the robot and its environment. In the test example, the planning scene contains the position and geometry of the Nyboda table, and soda can as well as information about the HSR. This environment is read from a file with a .scene file format. The HSR dynamics and kinematic properties are read from URDF and SRDF files. (Unified Robot Description Format and Semantic Robot Description Format)[probably some reference to the file formats]. The base trajectory for the drive-by is also specified here since it defines the robot's base

position in the scene. This input base trajectory acts as a base position and orientation constraint.

The motion solver is set as AICO since the purpose of this project is to use the motion solver implementation instead of benchmarking a new solver. It is important to note that AICO does not guarantee that constraints are satisfied. Instead, it optimizes the trajectory according to a quadratic cost function, in which each constraint is weighted according to its cost.

The planning problem defines the type of problem being solved and the tasks to be completed. The problem definition used for the grasping phase is an Unconstrained Time-Indexed Problem since it defines a 'defines a problem minimizing the quadratic cost over a trajectory using a kinematic model of the system', which can be solved by AICO. The planning problem uses task maps, which are functions that map the configuration space to task space. EXOTica includes several common taskmaps that are utilized in this project to define constraints on the HSR during the grasping phase.

5.4.1 Taskmaps

Task maps are active at various points of the extended grasping phase depending on the sub-phase. A task map is considered active when its associated cost is greater than zero. Throughout the entire extended grasping phase, 2 task maps are constantly active:

- A Joint Limit task map penalizes joint limit violations to promote HSR arm trajectories that fall within viable configurations.
- A Base Position task map penalizes deviations from the drive-by base trajectory specified in the planning scene.

3 task maps are activated during the sub-grasping phase. Despite the notion that the task maps are active at a certain time, AICO attempts to optimize the trajectory such that the constraint imposed by the task is satisfied in the window of time that the task map is active.

- An End Effector Position Task map causes the HSR to reach the location of the target soda can object. This task map is activated in the window of the sub-grasping phase. The sub-grasping window is defined by

$$grasp\ window = t_{grasp\ start} : t_{grasp\ start} + grasp\ duration$$

where $t_{grasp\ start}$ is the point in time where the HSR is sufficiently close to reaching the soda can and *grasp duration* is the maximum time necessary to close the HSR gripper, which is 0.4 seconds.

- An End Effector Axis Alignment task map aligns the HSR gripper with the axis of the cylindrical soda can so the gripper stays around the soda can while the fingers of the gripper close. This Axis Alignment task begins at $t_{grasp\ start}$ and remains active to keep the soda can upright for the remainder of the extended grasping phase. [Insert figure]

- A Point to Plane task map named 'Lift Off Table' keeps the end effector a set distance above the plane of the table to avoid the arm and gripper colliding with the table. This task map is active momentarily before *grasp window* and momentarily after *grasp window* to position the gripper above the table when transitioning from sub-reach to sub-grasp and from sub-grasp to sub-leave, such that the gripper clears the end of the table when lowering the starting arm configuration. This task map results in a swooping motion of the gripper during the sub-grasping phase. This Lift-Off Table task map takes the place of collision checking the trajectory by creating a trajectory that likely does not collide.

5.4.2 Commentary on Task Map Weights

[create a diagram of extended grasping phase and when constraints are active.]

5.5 HSR Controller

The AICO solver outputs a trajectory with position waypoints equally spaced in time. The trajectory is specified through a joint trajectory controller in ROS and sent to the HSR. According to the ROS wiki documentation for the joint trajectory controller, a trajectory composed solely of position waypoints is linearly interpolated and results in a trajectory with discontinuous velocities at the waypoints.

Taking the difference in positions and dividing by the time step produces an array of desired velocities at each waypoint. This results in a cubic spline that ensures continuity at the velocity level. [Include a bit about simple action client and how trajectories get passed to HSR]

5.6 Procedure

5.6.1 Hardware and Simulation Specifications

drive-by pick-and-place motions are run on an HSR simulated in Gazebo. [Computer specs like in ***].

5.6.2 Grasping Phase

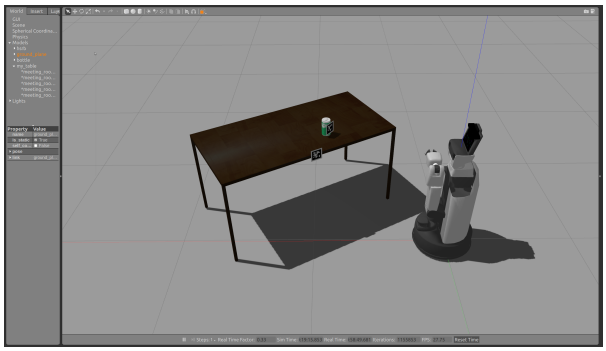
The drive-by base trajectory was defined in the planning scene as a 1.4-meter path parallel to the edge of the table with a duration of 10 seconds and a distance from the edge of the table of 0.35 meters. This trajectory was defined by two waypoints and results in a linear base trajectory that runs parallel to the Nyboda table. [Insert figure of a top-down view of setup in notes]. Since the task maps passed to the AICO solver do not depend on time, the 10-second duration can be scaled in time by increasing or reducing the time step between trajectory waypoints of the solved trajectory.

Due to limitations in the reachability of the HSR arm, the target object was placed near the edge of the table to minimize the likelihood of collision with the table. [Section Reference] further details the relationship between table height and bottle distance from table edge for the HSR.

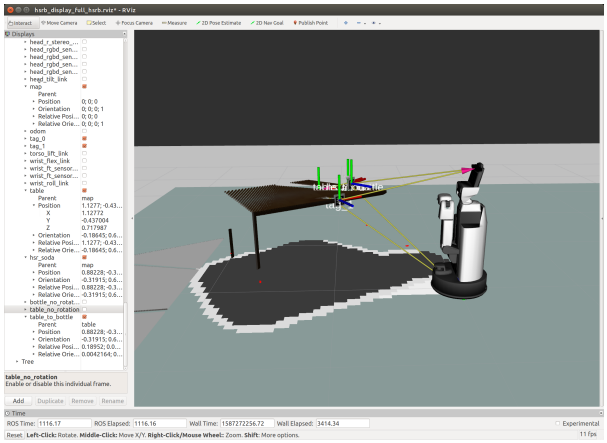
5.6.3 Object Detection

Visual markers are used as a fiducial to accurately acquire the position of the table and soda can using computer vision. The HSR includes AR marker detection which uses the depth sensor as well as the stereoscopic camera to determine the position of a defined object. However, AprilTags were used instead since AprilTags were found to permit tag detection from a longer range compared to AR markers. [20]..

Once the frames of both the table and soda can were determined, a series of transforms using Kinematics and Dynamics Library (KDL)[21] wrapped in EXOTica produced the absolute position and orientation of the table as well as the position of the bottle in the frame of the table.[references] Since the position of the soda can with respect to the table's frame is known, the planning scene utilized in the test example can be used to compute the extended grasping trajectory. The absolute position of the table allows the HSR motion planner to reach the start of the extended grasping phase, which is defined in the frame of the table. [Insert Image of tf's in rviz]



(c) Gazebo Visualization



(d) Rviz visualization

Figure 12: Will clean up image. (a) Gazebo visualization of world. (b) Soda can and starting pose of HSR in the extended grasping phase in the frame of the table

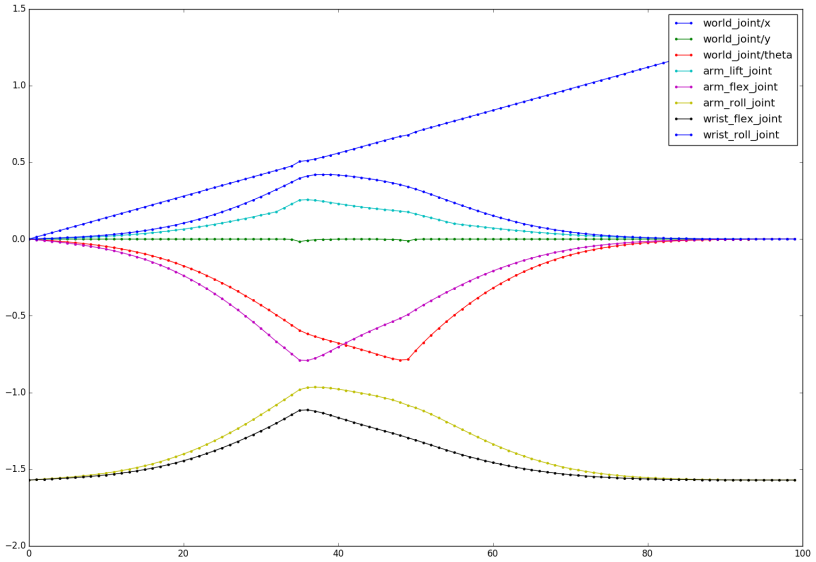
6 Results and Discussion

6.1 Trajectory Analysis

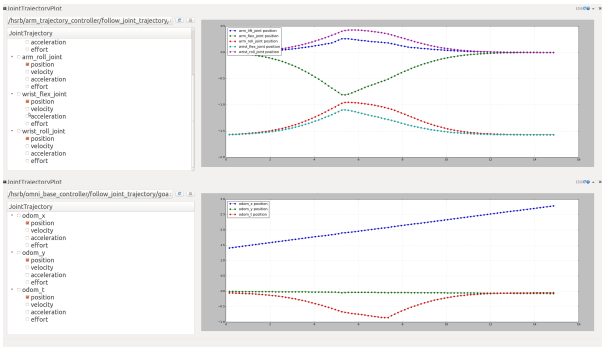
AICO outputs a trajectory that is smooth in position fig. 13(b), but there are no guarentees of smoothness in velocity or acceleration. For the extended grasping phase, AICO produced a trajectory with 100 waypoints. Initial trials with a timestep of 0.1 seconds for a 10 second extended grasping phase duration, found that the HSR could not reach waypoints in the alloted time and returned an error. The issue resulted from exceeding velocity constraints, specifically the arm-lift-joint (fig. 7(a)), which has a upper velocity limit of 0.2, as defined in the HSR URDF file. section 6.1 shows that the 15 second trajectory keeps the arm-lift-joint under the threshold. fig. 13 depicts the ideal trajectory output by the AICO solver, sent to the HSR through the joint trajectory controller.

6.2 Analysis of Plots

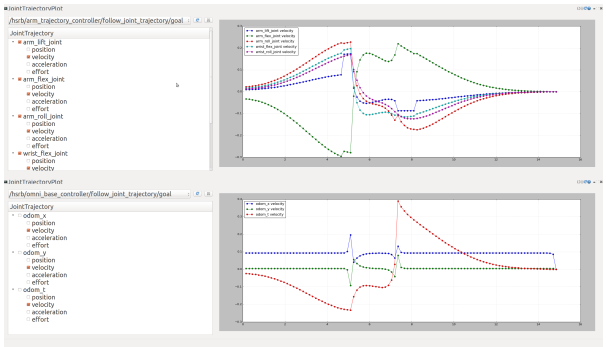
It is important to note that trajectory feedback measurements differ from the ideal trajectory due to noise in the robot system and also due to alterations made by the joint trajectory controller. For planning the path of the base, AICO gives a trajectory with x,y coordinate positions and an angle θ that represents the yaw of the HSR. The HSR base trajectory controller takes the x,y, θ values and converts the values into goals for the drive wheels and the base roll joint, described



(a) AICO trajectory output



(b) Waypoint Positions



(c) Waypoint Velocities

Figure 13: Ideal position and velocity values at each waypoint. (a) shows the joint position values output by the AICO solver as a sanity check for (b). (c) shows a plot of joint velocities at each waypoint. Values plotted from (b) and (c) were received from messages published by /follow_joint_trajectory/goal topics for the HSR arm and base. This topic publishes joint_trajectory messages at 10hz, which represent the ideal waypoints for the HSR to follow. Connections between points should be ignored since they are an artifact of plotting with RQT

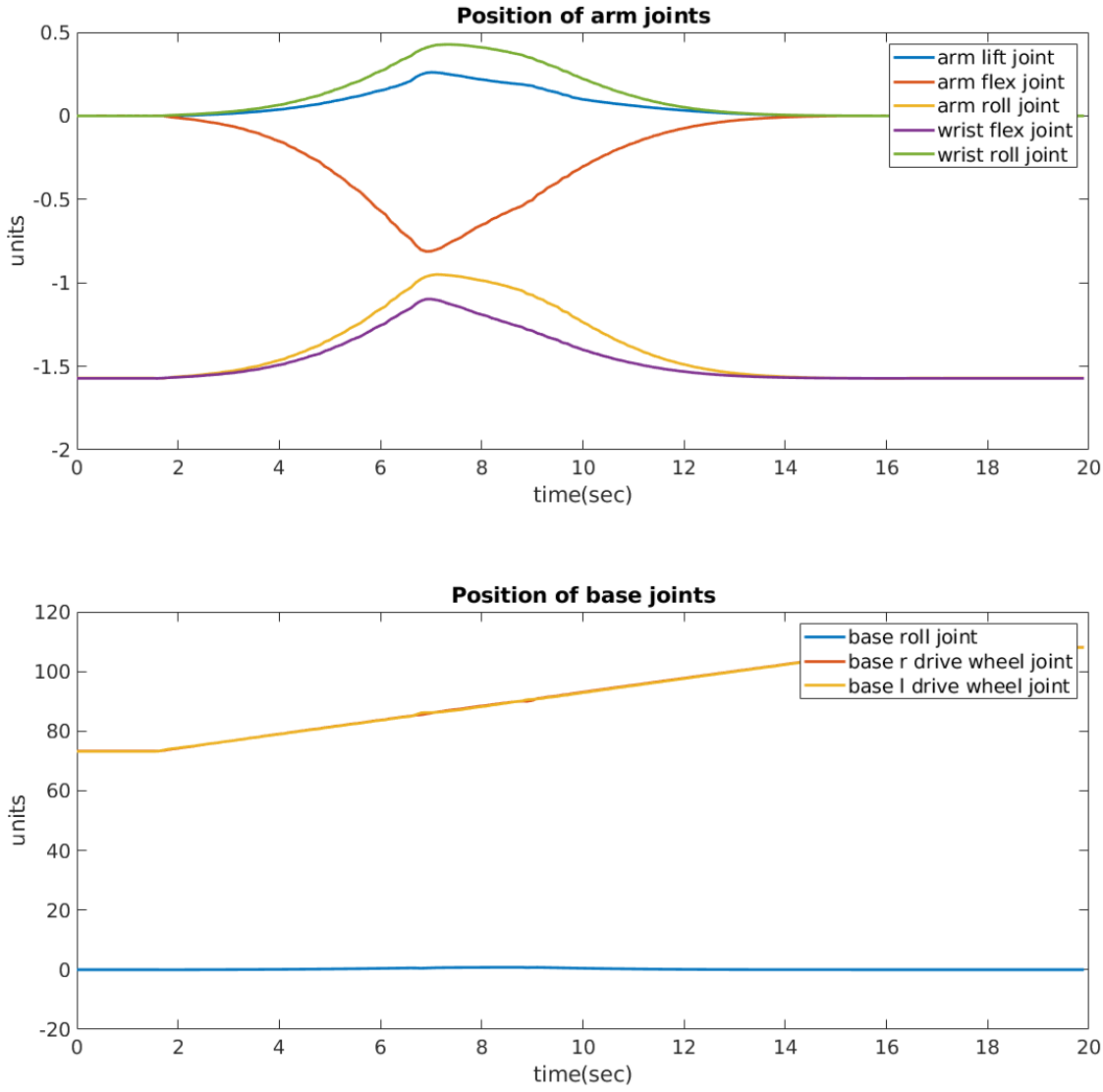


Figure 14: Graph of position of a trajectory executed by the HSR when position and velocities are specified. Trajectory starts at 1.5 seconds and ends at 16.5 seconds. Plots depict trajectory messages over the /hsrb/robot_state/joint_states, which publishes at a rate of 10 hz. This topic Connections between points should be ignored since they are an artifact of plotting with RQT

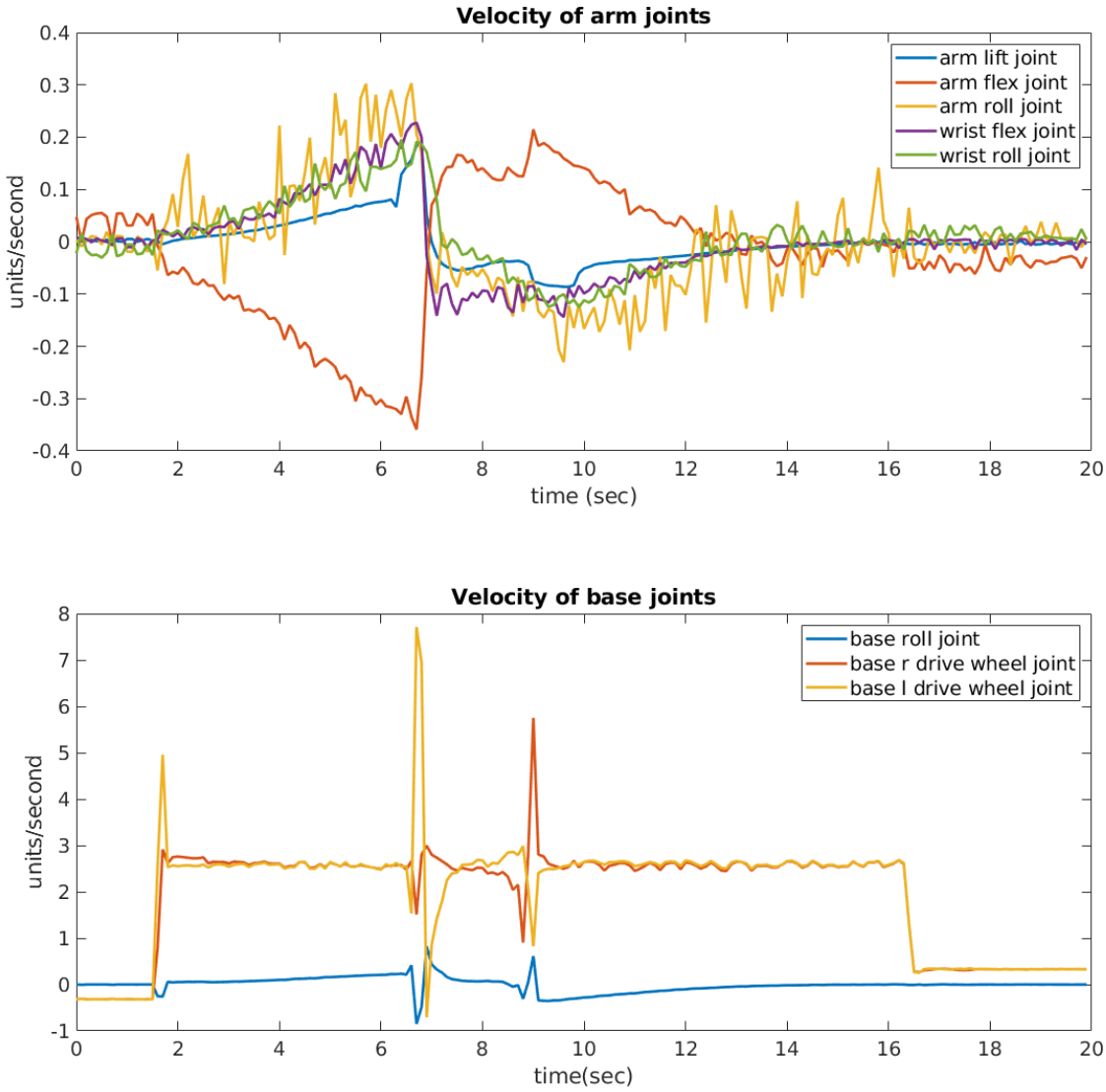


Figure 15: Graph of velocity of a trajectory executed by the HSR when position and velocities are specified. Trajectory starts at 1.5 seconds and ends at 16.5 seconds. Plots depict trajectory messages over the /hsrb/robot_state/joint_states, which publishes at a rate of 10 hz. This topic Connections between points should be ignored since they are an artifact of plotting with RQT

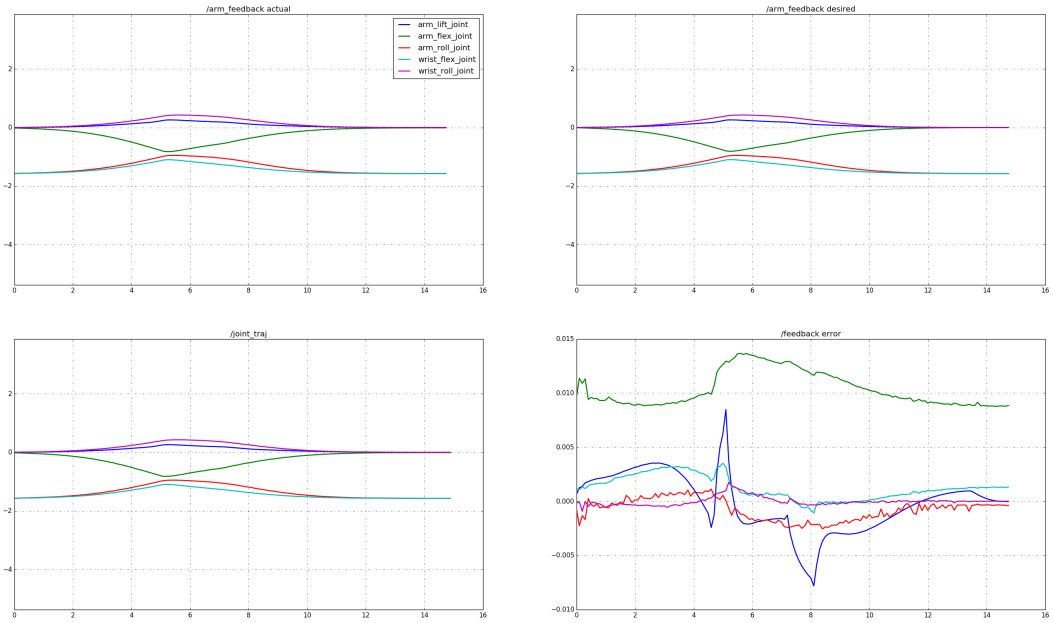


Figure 16: All trajectories were sampled from the simulated HSR at 10 Hz. The horizontal axis has units of time in seconds and the vertical axis has units of position, meters in the case of arm lift joint and radians for all else. (top left): actual joint positions of the sim HSR arm. (top right): Desired joint positions. (bottom right): the difference between position values in measured and desired joint positions. Actual and desired values were received from a control_msgs/FollowJointTrajectoryActionFeedback message, so the values have matching timestamps.

in [hsr section reference]. As a result, fig. 13 base joint trajectories and labels differ from the representation in section 6.1. Because base joint velocity, acceleration, and efforts are defined for the physical HSR joints, the latter labels are used for analyzing the trajectory.

6.2.1 Joint Positions

At the position level, the measured arm joint positions in section 6.1 match the ideal arm joint positions in fig. 13(a) and fig. 13(b). fig. 16 depicts that the error in position is small in comparison to the position values. Notably, the arm flex joint and arm lift joint deviate the most from the desired values. However, considering that the 2 links are at the base of the HSR arm kinematic tree and are responsible for moving the entire weight of the arm, relatively larger deviations are

acceptable.

Further, a visual check of the trajectory in Rviz and Gazebo confirms that the error in joint positions did not significantly impact the trajectory since the HSR successfully grasped the soda can during the extended grasping phase and reached the end pose.

From section 6.1 it appears that the HSR base follows the linear drive-by trajectory very closely, with slight deviations at the start and end of the sub-grasping phase between 7 and 9 seconds in section 6.1 indicated by small disturbances in base drive joint plots. These abrupt changes in position are due to task maps becoming active during the sub-grasping phase. The base roll joint plot does not accurately capture the change in HSR body orientation, due to the large values in the drive wheel joint position. [insert new plots without resets and separate roll joint from drive joints].

The HSR accomplished the grasping phase of the drive-by pick-and-place problem in simulation. Although the joint position limits for the simulated HSR are slightly greater than the values given for the actual robot, the joint positions in the trajectory simulation fall below both boundaries as a result of the joint limit task map. [insert chart of joint limits in hsr manual compared to limits in urdf file.]

6.2.2 Joint Velocities

[insert table of joint velocity limits] The joint velocity measurements fall within velocity limits for both the physical robot and simulated robot. [reference limit table].

One issue is the prevalence of periodic perturbations in the desired arm joint velocities in fig. 17 that are not present in the desired velocity plot in fig. 13(c). The joint velocity values in fig. 13(c) represent the velocity goal at each trajectory waypoint and are obtained from differentiating the joint position trajectory generated by the AICO solver. The perturbations in fig. 17 result from interpolation performed by the HSR trajectory controller that ensures smoothness in velocity.

Another glaring issue is the significant noise in the measured velocities in fig. 17. The histogram plot in fig. 17 appears to be zero-mean and roughly gaussian. Thus, three possibilities were proposed to explain the disturbances.

- 1. The deviations are due to noise in measurement.
- 2. The deviations are due to noise in the simulated joint velocity.

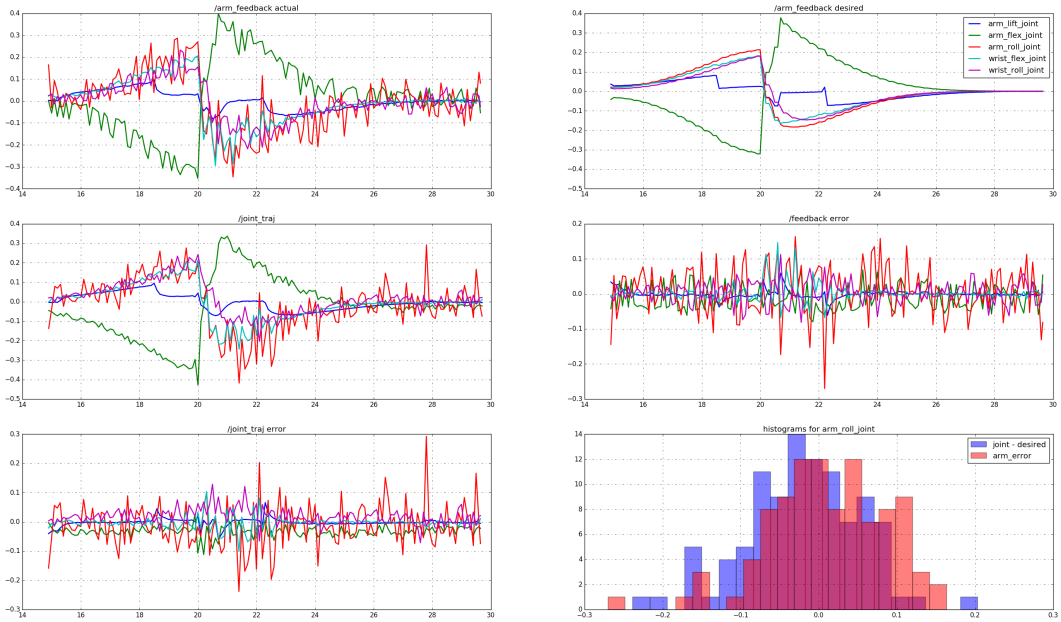


Figure 17: For non-histogram plots, the horizontal axis has units of seconds and the vertical axis has units of velocity. (Top row): Measured and desired joint velocity values for the simulated HSR. Corresponding points have matching timestamps. (Middle left): Velocity measurements sampled from the simulated HSR trajectory. Timestamps do not exactly match those of the desired plot. (Middle Right): Plot of difference between time-synchronized joint velocity measurements. (Bottom Left): Plot of difference between velocity measurements of /joint_traj and desired. (Bottom right): histogram of velocity error values for the arm roll joint.

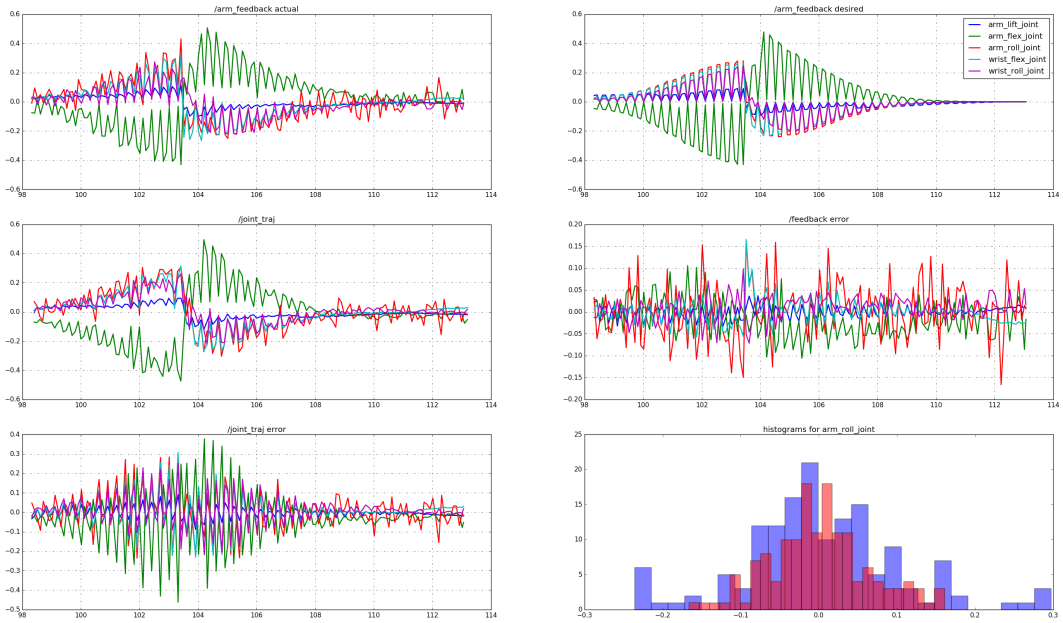


Figure 18: Plots in a similar order to those in fig. 17. Joint velocities were set to zero at each trajectory waypoint.

- 3. The velocity is attempting to return to zero at each time step, resulting in large disturbances.

fig. 18 demonstrates the effect of setting the velocity at each waypoint to zero. The perturbations are much more ordered than in the case of fig. 17, and the desired joint velocity plot would show more extreme rapid changes in velocity. This rules out explanation 3.

Since the physics are simulated, explanations 1 and 2 become indistinguishable unless noise is specifically added after a measurement is taken. Conditions relating to noise must be described in the HSR urdf. Checking the urdf file yields Gaussian noise tags for joints in the head and base segments of the HSR, but none specified for arm joints. However, a damping parameter in the dynamics tag is specified for each arm joint. The damping parameter is set to 1.0 for all arm joints aside from the arm roll joint which has a damping parameter of 0.1. The dynamics tag specifies physical properties of joints and the damping parameter specifically controls the decay of velocity. The fact that the damping parameter of the arm roll joint is lower partially explains the higher amounts of noise in fig. 17.

6.2.3 Joint Acceleration

Since the joint velocity values for the drive-by pick-and-place trajectory are not smooth, the joint acceleration plot exhibits high peaks that correspond with sharp velocity changes. section 6.2.3 depicts the desired acceleration for the trajectory from the AICO solver. The normalized desired acceleration plot shows that acceleration limits are exceeded for the arm-lift and arm-flex joints at approximately 5 seconds. The two peaks coincide with the start and end of the end-effector positioning and axis alignment tasks. This indicates that further modifications to the drive-by pick-and-place problem targets and costs would be required to drastically decrease the magnitude of the acceleration peaks. In addition, reducing the joint velocities by executing the trajectory over a longer duration of time would also contribute to decreasing the joint acceleration values.

The acceleration values calculated from the simulated trajectory in section 6.2.3 also display two peaks at the 5-second mark that coincide with the peaks in the desired acceleration plot. It is also observed that the arm-roll-joint acceleration values have noise that is larger in magnitude compared to the noise in other arm joints, which likely results from the damping parameter of the simulated joint described earlier. It is important to note that the acceleration limits for the HSR used in the normalization plots correspond to limits of the physical HSR and are not accounted for in the URDF file of the simulated robot. Further testing on the actual HSR is needed to verify and address the issue of exceeding acceleration limits.

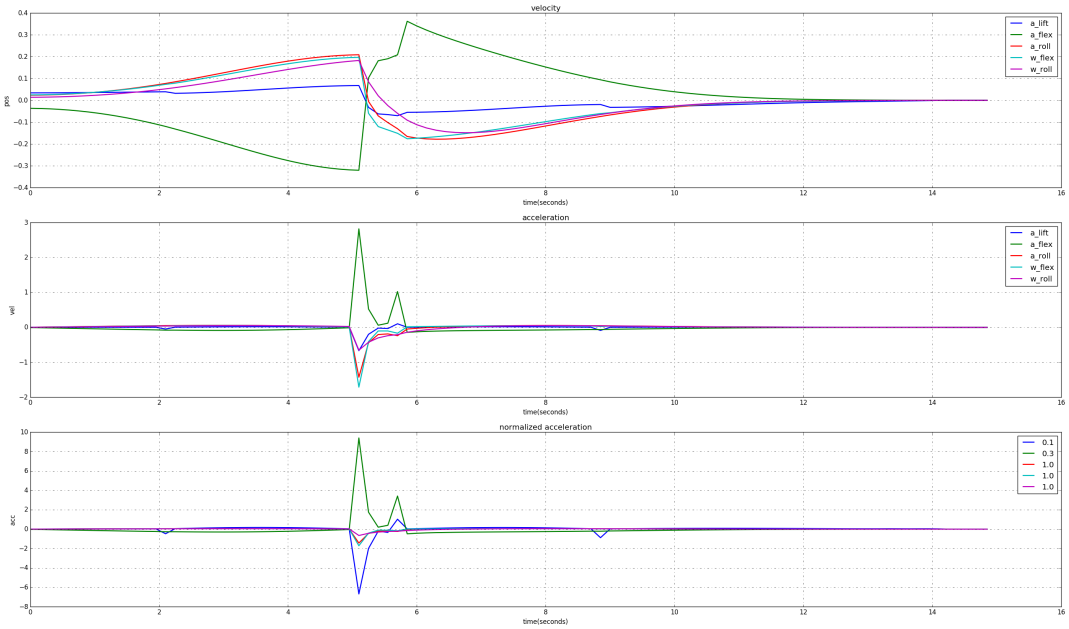


Figure 19: (Top): Desired velocity values from fig. 13(c). (Middle): Desired acceleration values calculated by taking the difference between adjacent velocity points and dividing by the time step. (Bottom): Desired joint acceleration values normalized by acceleration limits, displayed in the legend.

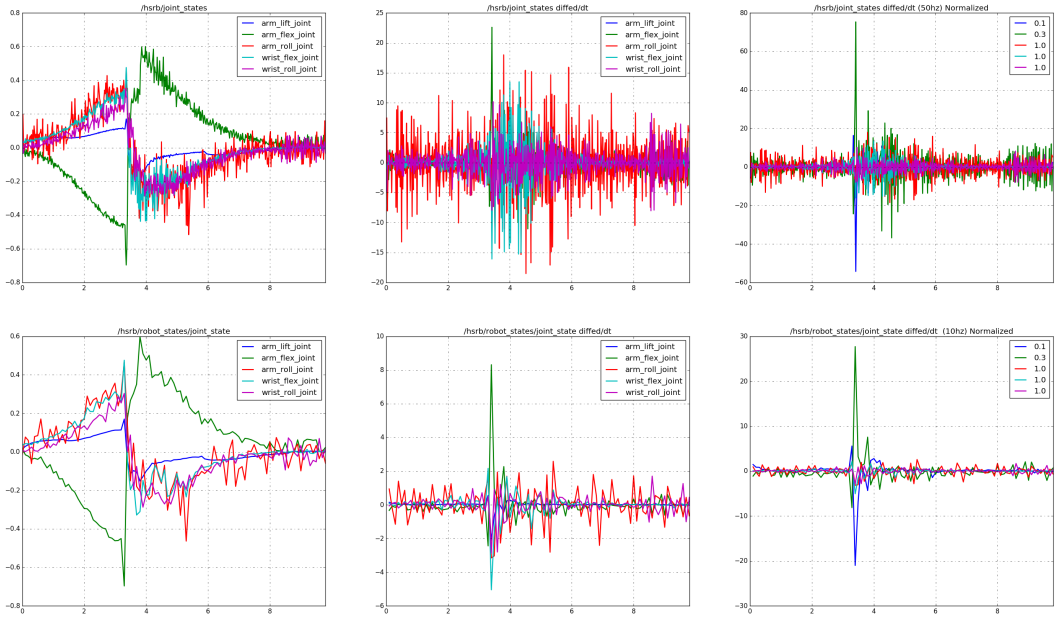


Figure 20: HSR arm joint accelerations obtained by taking the differentiating velocity with respect to the timestep. (Top row): data from /hsrb/joint-states ROS topic, which receives data from Gazebo at 50 Hz. (Bottom row): data from hsrb/robot-states/joint-state, which subsamples the data from /hsrb/joint-states through the HSR joint-state-publisher at 10 Hz. (First column): Velocity published from the respective topics. (Second column): Acceleration values calculated by taking the difference between adjacent velocity points and dividing by the time step. (Third column): Joint acceleration values normalized by acceleration limits, displayed in the legend.

7 Conclusion

This paper introduces the drive-by pick-and-place method and implements it on the Toyota HSR to accomplish the pick-and-place task in a simplified environment. It is demonstrated that the whole-body drive-by planning method implemented on the HSR succeeds in accomplishing the pick-and-place task and generates trajectories that are smooth in position through the use of AICO for planning an extended grasping phase.

This method in its current implementation has certain limitations. The trajectory has not yet been executed on the physical HSR, so differences in the magnitude of noise on joint measurements and measured joint accelerations may differ from the values in simulation.

The predefined drive-by base trajectory is linear and assumed to be given prior to planning. Moreover, the start time and duration of the sub-grasping phase of the grasping step is pre-specified. This results in the HSR having to drive to a designated start pose to begin the grasping phase and assumes that a 1.4-meter path parallel to the table edge is collision-free. Further work would incorporate an additional path planning step to identify a base trajectory for the grasping step that is collision-free and brings the robot sufficiently close to the grasping target for the duration necessary to grasp the object. Identifying the earliest base pose that is within reaching distance to the target object could aid in determining the optimal start time for the sub-grasping step.

This project only plans for grasping a soda can. The HSR gripper is capable of grasping larger objects, so gripping duration and the distance to separate gripper fingers should also be parameters for the algorithm to determine.

Using fiducial markers allows objects to be easily identified using computer vision. The HSR has an Nvidia Jetson TK1 installed as an auxiliary computer to enable GPU computing. This allows the HSR to perform objection recognition using neural networks with algorithms such as YOLO. Running YOLO and training new models on the HSR is outlined in the HSR documentation. [3].

This project plans motion in a static environment. However, EXOTica allows for planning in a dynamic environment as demonstrated in [15]. Extending the method to a time-varying environment would increase the HSR's utility as a service robot for human interaction in real-world scenarios.

8 Bibliography

References

[1] "RoboCup@Home Rules and Regulations," p. 72, 2020.

[2] S. M. LaValle, *Planning Algorithms*. Cambridge: Cambridge University Press, 2006.

[3] *HSRB_Manual*. Toyota Motor Corporation.

[4] M. Wada, A. Takagi, and S. Mori, "Caster drive mechanisms for holonomic and omnidirectional mobile platforms with no over constraint," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, (San Francisco, CA, USA), pp. 1531–1538, IEEE, 2000.

[5] P. Newman, "C18 — Mobile Robotics," Oct. 2017.

[6] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," 2009.

[7] P. Kaiser, M. Grotz, E. E. Aksoy, M. Do, N. Vahrenkamp, and T. Asfour, "Validation of whole-body loco-manipulation affordances for pushability and liftability," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, (Seoul, South Korea), pp. 920–927, IEEE, Nov. 2015.

[8] S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," tech. rep., 1998.

[9] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, Aug. 1996. Conference Name: IEEE Transactions on Robotics and Automation.

[10] J. Gammell, "C18 Lecture Slides."

[11] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on*

Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), vol. 2, pp. 995–1001 vol.2, Apr. 2000. ISSN: 1050-4729.

[12] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *2009 IEEE International Conference on Robotics and Automation*, (Kobe), pp. 625–632, IEEE, May 2009.

[13] D. Berenson, J. Chestnutt, S. S. Srinivasa, J. J. Kuffner, and S. Kagami, "Pose-constrained whole-body planning using Task Space Region Chains," in *2009 9th IEEE-RAS International Conference on Humanoid Robots*, (Paris, France), pp. 181–187, IEEE, Dec. 2009.

[14] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, "Development of Human Support Robot as the research platform of a domestic mobile manipulator," *ROBOMECH Journal*, vol. 6, p. 4, Dec. 2019.

[15] Y. Yang, W. Merkt, V. Ivan, and S. Vijayakumar, "Planning in Time-Configuration Space for Efficient Pick-and-Place in Non-Static Environments with Temporal Constraints," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pp. 1–9, Nov. 2018. ISSN: 2164-0580.

[16] M. Toussaint, "Robot trajectory optimization using approximate inference," in *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, (Montreal, Quebec, Canada), pp. 1–8, ACM Press, 2009.

[17] E. Todorov and Weiwei Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference, 2005.*, (Portland, OR, USA), pp. 300–306, IEEE, 2005.

[18] T. P. Minka, "Expectation Propagation for Approximate Bayesian Inference," p. 8.

[19] V. Ivan, Y. Yang, W. Merkt, M. P. Camilleri, and S. Vijayakumar, "EXOTica: An Extensible Optimization Toolset for Prototyping and Benchmarking Motion Planning and Control," in *Robot Operating System (ROS)* (A. Koubaa, ed.), vol. 778, pp. 211–240, Cham: Springer International Publishing, 2019. Series Title: Studies in Computational Intelligence.

[20] J. Wang and E. Olson, "AprilTag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Daejeon, South Korea), pp. 4193–4198, IEEE, Oct. 2016.

[21] "Kinematics and Dynamics Library (KDL)."