UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS num. 000

From Global to Local: Maintaining Accurate Mobile Manipulator State Estimates Over Long Trajectories

Filip Maric

Zagreb, September 2017.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Kako biste uklonili ovu stranicu, obrišite naredbu `\izvornik`.

dsadsa

CONTENTS

1. Introduction

In applications ranging from military operations to extraterrestrial exploration, mobile manipulators present themselves as the ideal answer to many challenges being addressed by robots. The mobile platform increases the workspace of a conventional manipulator while also enabling optimal positioning. The increased workspace also provides the capability of performing industrial tasks that would otherwise require multiple manipulators. A comprehensive survey on mobile manipulator systems can be found in (Bloch et al., 2003).

The mobile manipulator can be described as a robotic system composed of a manipulator arm mounted on top of a mobile platform. A variety of issues related to mobile manipulators have been explored in the past two decades. These include dynamic and static stability, force development and application, maximum payload determination, etc (Papadopoulos and Gonthier, 1999; Korayem and Ghariblu, 2004). However, a majority of the research concerning mobile manipulators deals with motion planning and state estimation (Yamamoto and Yun, 1992; Korayem et al., 2012).

This thesis explores the problems of state estimation and motion planning in the context of an omni-directional mobile manipulator. Chapter 2 introduces the "Thing" mobile manipulator used in this work. Each major component is described in appropriate detail, keeping in mind the context of research presented.

Chapter 3 deals with the challenge of state estimation for a mobile manipulator. Assuming accurate mobile base pose data, it is possible to estimate the state of the manipulator to a reasonable degree of accuracy using only a forward kinematics model described in section 3.2. The primary interest thus lies in obtaining the end effector pose estimate, as other estimates can be derived from it if necessary. Finding the base position relative to the starting position is explored in section 3.3 by fusing local odometry estimates in a probabilistic framework.

Finally, two approaches two motion planning are described in chapter 4. A classical approach enables defining priorities for kinematics tasks assigned to the robot, which are then applied in a classical optimization scheme. The more contemporary *sequential convex optimization* is also explored, which enables defining non-linear inequality and equality constraints. These two approaches offer a powerful framework for local motion planning and

trajectory optimization.

2. Thing mobile manipulator

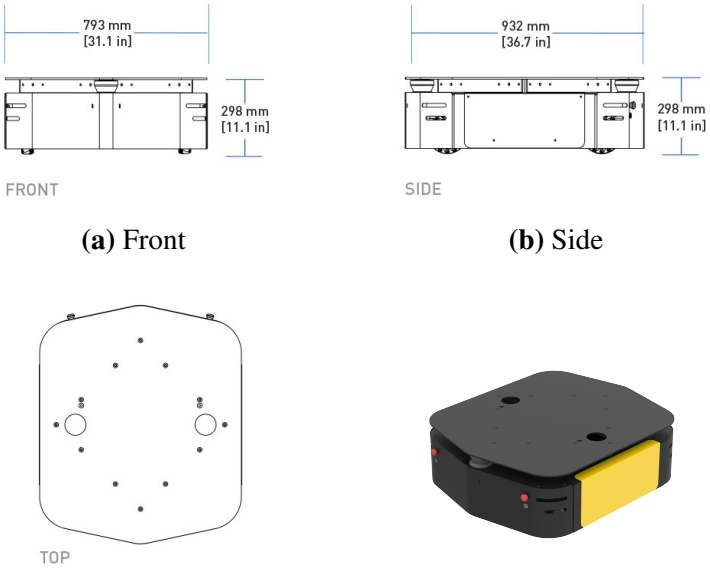
The Thing mobile manipulator is composed of a Ridgeback omni-directional platform and a UR10 manipulator arm equipped with a Robotiq 3-Finger Adaptive Robot Gripper is used as the end-effector. The three systems are essentially separate on the lower level: the RIdgeback, UR10 and Robotiq gripper local hardware interfaces are electronically independent self-contained systems. These local interfaces are unified on a higher level by the Ridgeback's computer operating as a server running the Robot Operating System (ROS). This

Figure 2.1: The Ridgeback mobile platform

platform is also equipped with a vast array of sensors (some not shown in **??**), making it ideal for autonomy research in robotics. The Ridgeback platform is outfitted with a Hokuyo laser range finder (LIDAR) and Kinect2 camera , giving valuable perception capabilities. The omni-directional drive enables field-of-view (FOV) control independent of both end-effector pose and base position, making it an interesting plaform for active perception research. The UR10 manipulatoris equipped with the advanced FT300 force-torque sensor and an Intel Realsense camera mounted on the gripper.

2.1. Ridgeback

Ridgeback is a midsize indoor robot platform that uses an omni-drive to move manipulators and heavy payloads with minimal constraints. The omnidirectional base provides precision positioning in constrained environments and comes fully integrated with onboard computer, front laser scanner and an IMU. Ridgeback offers native ROS and Gazebo integration and is compatible with a wide range of robot accessories. It is primarily designed for warehouse-



(a) Front **(b)** Side

(c) Top **(d)** Render

Figure 2.2: The Ridgeback mobile platform

ing applications as it is highly flexible in a controlled flat environment. The built in wheel odometry provides accurate local position estimates, with drift accumulating very slowly. For improved localization and obstacle avoidance, the navigation stack ROS package provides SLAM capability using the on-board LIDAR and IMU.

2.1.1. Hokuyo LIDAR

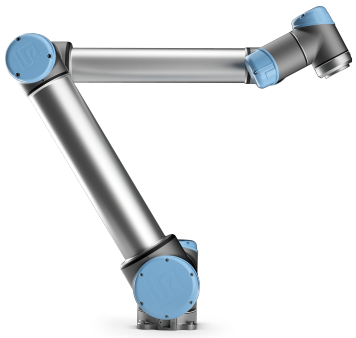
The Hokuyo UST-10LX Scanning Laser Rangefinder is a small, accurate, high-speed device for obstacle detection and localization of autonomous robots and automated material handling systems. This model uses Ethernet interface for communication and can obtain measurement data in a wide field of view up to a distance of 10 meters with millimeter resolution. This sensor uses a laser source to scan 270° field of view. Positions of objects in the range are calculated with step angle and distance. Sensor outputs these data through communication channel.

Table 2.1: UR10 specifications

Weight	28.9 kg
Payload	10 kg
Speed	Base and Shoulder joints: Max 120°/s All other joints: Max 180°/s Tool: approx 1 m/s
Repeatability	± 0.01 mm
Power consumption	average 350 W

2.2. UR10

The UR10 is Universal robotics largest collaborative industrial manipulator. It is a full 6DOF robotic arm capable of mimicking human arm movements. During operation it can handle weights up to 10 kg, making it viable for a wide range of tasks. The provided drivers enable performing collaborative tasks with human operators, allowing for easy positioning of the arm through compliant movement and preventing use of excessive force on impact.wwwww The data in table 2.1 clearly shows that the UR10 manipulator is meant for tasks involving



(a) Side

Figure 2.3: The UR10 manipulator

high precision and light payloads. The repeatability value of 0.01 mm shows that the UR10 can reach the same pose multiple times with a high degree of accuracy, which is important in industrial applications. Additionally, it's light weight and high end-effector modularity make it ideal for research purposes.

2.2.1. FT300

The UR10 on the Thing mobile manipulator has a variety of sensors built-in for use in various tasks. One such sensor is the FT300 force-torque sensor by Robotiq. In terms of mechanical fit, the Sensor has an embedded coupling to fit directly on the UR wrist. The tool side of the Sensor matches the UR bolt pattern. It-s application range from impedance control to perception and contact calibration of the manipulator.

(a) Front **(b)** Side

Figure 2.4: The UR10 manipulator

2.2.2. Robotiq Gripper

Figure 2.5: The Robotiq gripper

The Robotiq gripper is a three-fingered adaptive gripper used as the end effector on the UR10 manipulator. It is created for applications in advanced manufacturing and robotics research. All the fingers can be independently position,velocity and force controlled and can exert forces from 50 to 60 N at the finger tips.

3. State estimation

State estimation is one of the most often encountered problems when dealing with autonomous systems, as most algorithms require accurate information on the current state of the system. This type of system can be represented using non-linear system described by a difference equation and observation model with additive noise [1]

$$x_k = f\left(x_{k-1}\right) + w_{k-1},\ x \in \mathbb{R}^n \tag{3.1a}$$

$$z_k = h\left(x_k\right) + v_k,\ z \in \mathbb{R}^k \tag{3.1b}$$

We define the estimation problem as finding the state estimate $\hat{x}_k$ given the measurements $z_0 \ldots z_k$ subject to measurement noise v and past states $x_0 \ldots x_{k-1}$ subject to process noise w. Various algorithms have been developed that use different mathematical frameworks to find the best estimate according to some optimality criterion (Thrun et al. (2005)).

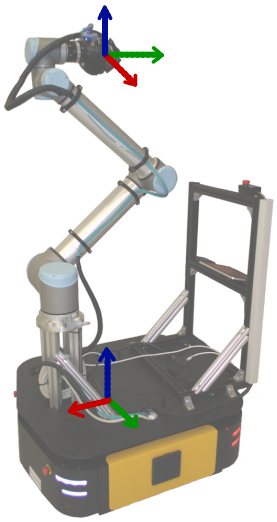


Figure 3.1: The Thing mobile manipulator and the frames subject to estimation

The problem of state estimation for a mobile manipulator is twofold: it is necessary to estimate the pose of the end-effector in the frame of the mobile base while also localizing the mobile base itself in the workspace. The pose of the end-effector in the base frame is

[1] we leave out separate expressions for input variables as they can be included within the functions f and h

estimated using a kinematic model. Joint position values are used to provide a pose estimate, ignoring various dynamic and environment effects. Estimating the base state (*localization*) is a more complex problem; the dynamic and environment effects on base movement significantly affect the steady-state. Consequently, in addition to kinematic models we have to rely on movement estimates from sensor data which is inherently noisy. The imperfect model and sensor data are fused in a single estimate using a probabilistic framework.

In this chapter, we first define our state vector of interest in section 3.1. Section 3.2 contains details on the derivation of the kinematic model used in end-effector pose estimation assuming sufficient knowledge of the base pose. The probabilistic framework used for estimating the base pose is described in section 3.3, where wheel and LIDAR odometry are fused to provide an accurate estimate. The focus of this chapter is estimating the pose of the end-effector and mobile base, but the methods are extendable to any frame on the platform.

3.1. State

Most high-level control and task algorithms require information on the pose of the mobile manipulator frames in Cartesian space. A frame pose represents complete informations about a frames position and orientation in a given reference frame. The pose is often represented by a *homogenous transformation matrix* T

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \tag{3.2}$$

The rotation matrix $R \in \mathbb{R}^{3x3}$ represents the orientation of the tool frame t as seen in the global frame 0, while translation is represented by the vector $p \in \mathbb{R}^3$. Estimating the matrix T for the end-effector is a 9-dimensional problem which can be further reduced. It is possible to fully define it the vector r consisting of the translation vector p and a *quaternion* q

$$r_{EE}\left(T_0^{EE}\right) = \begin{bmatrix} x & y & z & q_w & q_x & q_y & q_z \end{bmatrix}^T \tag{3.3}$$

We can reduce this even further for the mobile base, as it moves on a 2D plane and thus only rotates alongside one axis and has a constant z coordinate

$$r_B\left(T_0^B\right) = \begin{bmatrix} x & y & \theta \end{bmatrix}^T \tag{3.4}$$

Our estimated state consists of end-effector and base poses comes in the form of a 10-dimensional *task vector* s

$$s = \begin{bmatrix} r_{EE} & r_B \end{bmatrix}^T \tag{3.5}$$

Using this representation, the state estimation problem has been reduced from 24 down to 10 dimensions. Reducing the dimensionality of the problem makes it computationally less expensive, which is vital for real-time operation.

3.2. Kinematic model

If we disregard the various dynamic and environment effects, the Thing can be modelled as a manipulator. The Ridgeback platform is system with holonomic constraints and it's global position and the orientation can be viewed as a chain of two prismatic and a rotational joint. These three joints make up the base upon which we add the kinematic model of the UR10 manipulator, which results in a system kinematically indistinguishable from a redundant manipulator.

The UR10 joint position measurements are reasonably accurate and thus this model can be used to produce a valid state estimate of the local end-effector pose. Assuming valid estimates for the base position and orientation, this model will accurately represent the true state of the mobile manipulator.

3.2.1. Forward kinematics

We define forward kinematics as a set of equations that express the pose of a certain frame on a kinematic chain as a function of configuration-space values q

$$q = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & \hat{x} & \hat{y} & \hat{\theta} \end{bmatrix}^T \quad . \tag{3.6}$$

The equations are commonly derived using the transforms defined in (3.2). The transform between two frames can be defined as a function of only four parameters using the Denavit-Hartenberg notation.

$$T_{k-1}^{k}(d,\ \theta,\ a,\ \alpha) = \begin{bmatrix} \cos\theta_k & -\cos\alpha_k \sin\theta_k & \sin\alpha_k \sin\theta_k & a_k \cos\theta_k \\ \sin\theta_k & \cos\alpha_k \cos\theta_k & -\sin\alpha_k \cos\theta_k & a_k \sin\theta_k \\ 0 & \sin\alpha_k & \cos\alpha_k & d_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad . \tag{3.7}$$

The parameters a, α are usually constant and represent the structure, while d, θ represent the prismatic and rotational joint values and biases. Using the DH parameters as defined in table A.1, we derive all the transforms (3.7) from global frame to the end effector frame. Multiplying these transforms gives us the transform from the global frame to any frame of the mobile manipulator

$$T_0^{EE} = \prod_{k=1}^{N_{EE}} T_{k-1}^{k} \quad . \tag{3.8}$$

The forward kinematics model is subject to various joint angle and link length biases, as it is possible the given DH parameters do not exactly correspond to the physical manipulator. However the estimate provided is still has an error significantly lower than the error resulting from base localization and thus can be disregarded in the current analysis.

It is also interesting to note that the forward kinematics equations are non-injective surjective functions. Every q corresponds to only one pose T, even though a given T might correspond to multiple values of q. The analytic formulation of the transform from the global frame to the end-effector can be found in the appendix.

3.2.2. Differential forward kinematics

Equations defined in 3.2.1 provide an estimate of the end-effector and base pose using the configuration vector q. However, algorithms used in motion planning and trajectory optimization require information on how configuration change affects the pose. This information is contained within the manipulator Jacobian J. We define the Jacobian J as the derivative of the function $f\left(x\right) : \mathbb{R}^n \rightarrow \mathbb{R}^k$ with respect to the vector z.

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & & \\ \frac{\partial f_k}{\partial x_1} & \frac{\partial f_k}{\partial x_2} & \cdots & \frac{\partial f_k}{\partial x_n} \end{bmatrix} \tag{3.9}$$

Now $\dot{f}$ can be calculated by multiplying the matrix with $\dot{x}$

$$\dot{f} = J\dot{x} \tag{3.10}$$

For a sufficiently small Δ, equation 3.10 can be extended to produce a first order approximation $\hat{f}_k$

$$\hat{f}_k = f_{k-1} + J\left(x_k - x_{k-1}\right) \tag{3.11}$$

This framework can be employed in various state estimation and motion planning algorithms.

End-effector Jacobian

The end-effector Jacobian is defined as the derivative of the end-effector pose 3.3 with respect to the configuration vector q. While it can be derived analytically, it is often simpler to derive it using the geometry of the chain. Here the Jacobian takes the form

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}, \tag{3.12}$$

where the matrix J_v is the position Jacobian and J_ω matrix is the orientation Jacobian. In this case, the components representing orientation change are equal to the rotation axes of the joint q_j

$$J_{\omega_j} = \frac{\partial o}{\partial q_j} = z_j\,. \tag{3.13}$$

The components representing position difference for rotational joints are

$$J_{v_j} = \frac{\partial p}{\partial q_j} = z_j \times (w - p_j)\,. \tag{3.14}$$

For prismatic joints the orientation component is $J_{\omega_j} = 0$, while the translational component lies on the z axis

$$J_{v_j} = \frac{\partial p}{\partial q_j} = z_j\,. \tag{3.15}$$

As seen in Algorithm 1, the geometric derivation is suitable for software implementation as it does not require symbolic calculations and relies only on matrix operations.

Algorithm 1: Geometric Jacobian calculation

Data: Kinematic chain transforms T_i and joint type flags rev_i

Result: End-effector Jacobian matrix J

$T = I$;
for $i = 1, 2, ..., N$ **do**
  $T = TT_i$;
  $R_i = R(T)$;
  $O_i = O(T)$;
  if $i = 1$ **then**
    $z_i = (rev_i)\mathbf{1}$;
    $v_i = (rev_i)z_i \times w + (1 - rev_i)z_i$;
  else
    $z_i = (rev_i)R_{i-1}\begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$;
    $v_i = (rev_i)z_i \times (r - O_{i-1}) + (1 - rev_i)z_i$;
  end
end
$J = \begin{bmatrix}v & z\end{bmatrix}^T$

The *twist* vector w is defined as the composition of translational and rotational velocities. For small enough pose difference Δr, the vectors are approximately equal

$$w = \begin{bmatrix} v \\ \omega \end{bmatrix} \approx \begin{bmatrix} \Delta p \\ \Delta o \end{bmatrix} = \Delta r\,. \tag{3.16}$$

Calculating the position difference is straightforward and comes down to subtracting the goal and current positions

$$\Delta p = p - p_{goal} \tag{3.17}$$

The orientation difference is calculated as follows (detailed derivation in appendix)

$$\Delta o = 0.5L^{-1}\left(n_c \times n_d + s_c \times s_d + a_c \times a_d\right) \tag{3.18a}$$

$$L = -0.5\left(\left[n_c\right]_\times \left[n_d\right]_\times + \left[s_c\right]_\times \left[s_d\right]_\times + \left[a_c\right]_\times \left[a_d\right]_\times\right) \tag{3.18b}$$

3.3. Localization

When considering the kinematic model sufficiently accurate base pose estimates x, y, θ are assumed. In reality, obtaining accurate pose estimates is non-trivial and often requires fusing sensor data while taking in to account the probabilistic nature of measurements.

Localization of the Thing mobile manipulator employs a motion model of the omni-directional drive on the Ridgeback and data provided by the LIDAR. The wheel and LIDAR odometry provide Cartesian velocity estimates with covariance data. This data is fused using an extended Kalman filter and the resulting estimate is more accurate than simply integrating odometry data.

3.3.1. Wheel odometry

In classical mechanics a system may be defined as *holonomic* if all constraints of the system are holonomic. For a constraint to be holonomic it must be expressible as a function:

$$f(x_1,\ x_2,\ x_3,\ \ldots,\ x_N,\ t) = 0 \tag{3.19}$$

i.e. a holonomic constraint depends only on the coordinates x_j and time t. It does not depend on the velocities or any higher order derivative with respect to time t. Decoupled translational and angular velocities make the estimation process less prone to error, as angular velocities are usually much more difficult to estimate. Having wheel encoder velocity readings w_0, w_1, w_2, w_3, a Cartesian velocity estimate for the base can be calculated

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \frac{R}{4}\begin{bmatrix}1 & 1 & 1 & 1\\ -1 & 1 & -1 & 1\\ -\frac{1}{k} & -\frac{1}{k} & \frac{1}{k} & \frac{1}{k}\end{bmatrix}\begin{bmatrix}w_0\\ w_1\\ w_2\\ w_3\end{bmatrix} \quad . \tag{3.20}$$

Integrating these velocities makes it possible to localize the base relative to the starting position. Odometry estimates are assumed to have a normal distribution, accounting for measurement noise and kinematic nature of the model (ignores dynamic effects)

$$r_B \sim \mathcal{N}(\mu_{r_B},\ \Sigma)\,. \tag{3.21}$$

The covariance matrix Σ in (3.21) is set to a constant value empirically found to sufficiently represent estimate uncertainty

$$\Sigma = diag\left(0.001, 0.001, 0.001, 100000, 100000, 0.03\right) . \tag{3.22}$$

3.3.2. LIDAR odometry

3.3.3. Extended Kalman filter

The wheel and LIDAR odometry data is fused using an Extended Kalman Filter [2](Moore and Stouch, 2014). A process and observation model is used to represent the movement of the Thing mobile manipulator:

$$x_k = f\left(x_{k-1}\right) + w_{k-1},\ x \in \mathbb{R}^n \tag{3.23a}$$

$$z_k = h\left(x_k\right) + v_k,\ z \in \mathbb{R}^k \tag{3.23b}$$

Without going in to the detail on the derivation of the EKF algorithm, it can be described as having two steps. The model forecast step uses the motion model and last estimated state $\hat{x}_k^+$ to produce an *a priori* estimate of the next state $\hat{x}_k^-$ and covariance P_k^-

$$\hat{x}_k^- = f\left(\hat{x}_{k-1}^+\right) \tag{3.24a}$$

$$P_k^- = J_f\left(\hat{x}_{k-1}^+\right) P_{k-1}^+ J_f^T\left(\hat{x}_{k-1}^+\right) + Q_{k-1} \tag{3.24b}$$

The correction step makes use of local sensor odometry to calculate the *Kalman gain* K_k and produce *a posteriori* estimates $\hat{x}_k^+$ and P_k^+

$$\hat{x}_k^+ \approx \hat{x}_k^- + K_k\left(z_k - h\left(\hat{x}_k^-\right)\right) \tag{3.25a}$$

$$P_k^+ = \left(I - K_k J_h\left(\hat{x}_k^-\right)\right) P_k^- \tag{3.25b}$$

$$K_k = P_k^- J_h^T\left(\hat{x}_k^-\right)\left(J_h\left(\hat{x}_k^-\right) P_k^- J_h^T\left(\hat{x}_k^-\right) + R_k\right)^{-1} \tag{3.25c}$$

The J_f and J_h matrices in (3.24) (3.25) represent the process and measurement model Jacobians, while the process and measurement noise covariance is represented by the Q and R matrices. Estimates provided by (3.24) (3.25) are first-order estimates, higher order estimators of this type exist but are computationally infeasible for real-time applications.

It is also important to note that the EKF assumes white Gaussian noise and thus can only be successfully applied when the actual process and measurement noise have a near-Gaussian distribution.

[2]EKF in rest of text

4. Motion planning

As mentioned in chapter 3, the primary motion planning objective consists of reaching the desired pose with the end-effector and mobile base. However, moving the mobile manipulator safely and efficiently in the workspace is more involved and significantly constrains the primary objective. For instance, it is important to maintain distance from obstacles both with the platform and the manipulator. Additionally, various other performance metrics exist which may need to be maximized or maintained above a certain value. These objectives and constraints can be formalized in a way that fits the motion planning problem.

In section 4.1 the *inverse kinematics* problem is considered for mobile manipulator. This involves finding an configuration q that corresponds to a given end-effector pose r. Section 4.2 explores the problem of utilizing the extra degrees of freedom of the system to optimize movement, also referred to as *redundancy resolution*. The remaining sections explore two different approaches to redundancy resolution: *task-priority* and s*equential convex optimization*. The classic task-priority algorithm is computationally efficient extension to the optimization approach for inverse kinematics, but also highlights the drawbacks of a purely linear optimization algorithm. A more contemporary approach of sequential convex optimization provides a more general and powerful optimization infrastructure that is highly modular, at the cost of computational efficiency.

4.1. Inverse kinematics

Regulating kinematic tasks that are configuration dependent reduces to the problem

$$f(q) = 0 \tag{4.1}$$

where $q \in \mathbb{R}^n$ is the configuration vector. The classical approach to inverse kinematics would have us find the inverse of the function f

$$q = f^{-1}(0) \tag{4.2}$$

However, this inverse only exists in simple and non-redundant configurations, otherwise the solutions are infinite. Even in cases where the configuration is non-redundant , the inverse

function is still sometimes impossible to derive. In order to solve this problem, an optimization approach which avoids dealing with complicated analytical expressions is employed.

4.1.1. Differential inverse kinematics

In an inverse differential kinematics scheme used for redundant kinematic chains, $f(q)$ is regulated to the target value using

$$\frac{\partial}{\partial q} f = -\lambda_f f(q), \lambda_f > 0 \tag{4.3}$$

The solution to 4.3 is often rank-deficient, so a least squares formulation is employed

$$\min_{\dot{q}} \frac{1}{2} \left\| \frac{\partial f}{\partial q} \dot{q} + \lambda_f f(q) \right\|^2, \lambda_f > 0 \tag{4.4}$$

Letting $J = \frac{\partial f}{\partial q}$ and $a = -\lambda_f f(q)$, the solution to 4.4 that minimizes the L_2 norm is given by finding the pseudoinverse of the A matrix

$$\dot{q} = J^{\dagger} a \tag{4.5}$$

4.1.2. The Jacobian pseudoinverse and numerical filtering

In (4.3)-(4.17), the pseudoinverse operation denoted by † is defined as

$$J^{\dagger} = J^T \left(J J^T \right)^{-1} \tag{4.6}$$

While still more numerically stable than an outright inverse , this method is also subject to *kinematic singularities* in the manipulator. These are configurations where a large configuration change is required for a given task as seen by the linearized system. The components of the task Jacobian $\frac{\partial f}{\partial q}$ then approach 0, resulting in loss of rank. Implementing the pseudoinverse in a numerically stable way is very important for robustness and non-trivial.

Damped psudoinverse

We often use the more numerically stable *damped pseudoinverse* (Chan and Lawrence, 1988)

$$J^{\#} = J^T \left(J J^T + \lambda I \right)^{-1}, \lambda \in \mathbb{R} \tag{4.7}$$

There are also various contemporary approaches to implementing the pseudoinverse (Chiaverini et al., 1994). In this work we use the *selectively damped pseudoinverse* (Buss and Kim, 2005), alongside with *singular value filtering* described in (Colomé and Torras, 2015).

Selectively damped pseudoinverse

The matrix pseudoinverse operation can be performed using an SVD decomposition of the Jacobian

$$J = \sum_i \sigma_i u_i v_i^T \tag{4.8}$$

$$J^T \left(JJ^T\right)^{-1} = \sum_i \frac{1}{\sigma_i} v_i u_i^T \tag{4.9}$$

The damped pseudoinverse would in this case have the form

$$J^T \left(JJ^T + \lambda I\right)^{-1} = \sum_i \frac{\sigma_i}{\sigma_i^2 + \lambda} v_i u_i^T \tag{4.10}$$

The *selectively damped pseudoinverse* (Buss and Kim, 2005) equates to choosing a specific value λ_i for each singular value σ_i

$$J^{\#} = \sum_i \frac{\sigma_i}{\sigma_i^2 + \lambda_i} v_i u_i^T \tag{4.11}$$

Singular value filtering

The singular values of a Jacobian determine it's stability with respect to inversion. Lower singular values indicate that the matrix is closer to being singular. A proposed approach to solving this problem prior to using a specific inversion technique is *singular value filtering* (Colomé and Torras, 2012). This technique sets the lowest singular value of a given jacobian matrix J to σ_0

$$h_{v,\sigma_0}(\sigma) = \frac{\sigma^3 + v\sigma^2 + 2\sigma + 2\sigma_0}{\sigma^2 + v\sigma + 2}\,. \tag{4.12}$$

In (4.12) the shape factor v determines the difference between modified and original singular values $h_{v,\sigma_0}(\sigma) - \sigma$, a higher v results in lower differences. This results in a new Jacobian matrix

$$\hat{J} = \sum_i h_{v,\sigma_0}(\sigma_i) u_i v_i^T \tag{4.13}$$

Authors report that combining this filtering method with any of the above pseudoinverse approaches results in higher stability.

4.2. Redundancy resolution

Given that the valid set of solutions to 4.4 is in an affine subspace $\mathcal{F} \in \mathbb{R}^n$, the minimization scheme for solving a lower priority set of tasks would be

$$\min_{\dot{q} \in \mathcal{F}} \frac{1}{2} \left\| \frac{\partial g}{\partial q}\dot{q} + \lambda_g f(q) \right\|^2, \lambda_g > 0 \tag{4.14}$$

The process of defining additional constraints for a redundant system, thus reducing the affine subspace $\mathcal{F}$ is called *redundancy resolution*. The general solution to (4.14), which minimizes the L_2 norm of the primary task while making redundant DOFs available to other inputs z has the form

$$\dot{q} = J^{\#}a + \left(I - J^{\#}J\right)z, z \in \mathbb{R}^n \tag{4.15}$$

The operator $\left(I - J^{\#}J\right)$ is an orthogonal projector, thus the effect of the control z will not affect the outcome in a way that cancels out the first term.

4.3. Task-prioity kinematics

In (Nakamura et al., 1987; Nakamura, 1990), *null-space optimization* is utilized to achieve execution of a secondary task B given a primary task A

$$\dot{q} = A^{\dagger}a + \left(I - A^{\dagger}A\right)\tilde{B}^{\dagger}\left(b - BA^{\dagger}a\right) + \left(I - A^{\dagger}A\right)\left(I - \tilde{B}^{\dagger}\tilde{B}\right)z\,, \tag{4.16a}$$

$$\tilde{B} = B\left(I - A^{\dagger}A\right)\,. \tag{4.16b}$$

This **task-priority** approach to inverse kinematics can be extended to any number of tasks (Slotine, 1991). The main weakness of task-priority lies in the fact that lower-priority tasks will often become infeasible, causing the occurrence of an *algorithmic singularity* (Baillieul, 1985). In (Chiaverini, 1997), a modified approach is introduced , which guarantees robustness

$$\dot{q} = A^{\dagger}a + \left(I - A^{\dagger}A\right)B^{\dagger}b + \left(I - \begin{pmatrix}A\\B\end{pmatrix}^{\dagger}\begin{pmatrix}A\\B\end{pmatrix}\right)z \tag{4.17}$$

Here the trade-off is lower accuracy and slower convergence because the expression is only an approximation of (4.16a).

4.3.1. Motion planning scheme

The motion planning scheme implemented using the task priority formulation is focused on simplicity and robustness rather than optimal movement. The primary task is reaching a goal pose with the end effector, which is vital for most mobile manipulator applications. The secondary task handles mobile base trajectory following. As mentioned in 4.3, this framework can be extended to any number of tasks, but the increasing numerical instability results in diminishing returns.

$$\Delta q \approx J_{EE}^{\dagger}\Delta r_{EE} + \left(I - J_{EE}^{\dagger}J_{EE}\right)J_{B}^{\dagger}\Delta r_{B} \tag{4.18}$$

4.3.2. Drawbacks

The main drawback to task-priority kinematics in (4.16a) and (4.17) is that they cannot natively handle nonlinear inequality constraints (Moe et al., 2016) and equality constraints. Having the ability to define this type of constraint in motion planning is important for more complex systems such as mobile manipulators.

There are currently no open-source implementations for task-priority schemes , as the parameters and structure of the scheme vary greatly depending on the tasks and kinematic chain structure. A more general definition of the problem as a non-linear constrained optimization problem would allow us to utilize the vast array of open-source solvers, implementations and solution schemes developed for a wider range of problems.

4.4. Sequential convex optimization

Performing tasks with a kinematic chain can also be formulated as a bounded non-linear optimization problem P. The non-linear objective function f is minimized subject to non-linear inequality and equality constraints g, h, while keeping the parameter vector x within bounds.

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & x \in [x_{min}, x_{max}] \\ & g(x) \leq C_g \\ & h(x) = C_h \end{aligned} \tag{4.19}$$

We can solve this problem iteratively, by repeatedly constructing and minimizing a convex approximation of the problem $\tilde{P}$ until constraints are satisfied. With $\tilde{f}$ being the quadratic approximation of the objective function f,

$$\tilde{f}\left(x\right) = f\left(x^k\right) + \nabla f\left(x^k\right)\left(x - x^k\right) + \frac{1}{2}\left(x - x^k\right)^T Hf\left(x^k\right)\left(x - x^k\right) \tag{4.20}$$

and $\tilde{g}, \tilde{h}$ being local affine approximations of constraint functions

$$\begin{aligned} \tilde{g}\left(x\right) &= g\left(x^k\right) + \nabla g\left(x^k\right)\left(x - x^k\right) \\ \tilde{h}\left(x\right) &= h\left(x^k\right) + \nabla h\left(x^k\right)\left(x - x^k\right) \end{aligned} \tag{4.21}$$

Using quadratic programming (QP) techniques, we solve $\tilde{P}$

$$\begin{aligned} \min_{x} \quad & \mathcal{L}\left(x, \mu\right) \\ \text{s.t.} \quad & x \in [x_{min}, x_{max}] \end{aligned} \tag{4.22}$$

Where $\mathcal{L}\left(x, \mu\right)$ represents the problem Lagrangian

$$\mathcal{L}(x, \mu) = \tilde{f}\left(x\right) + \mu \sum_{i=1}^{i_{max}} \tilde{g}_i\left(x\right) + \mu \sum_{j=1}^{j_{max}} \tilde{h}_j\left(x\right) \tag{4.23}$$

The Lagrange multiplier $\mu > 0 \in \mathbb{R}$ is a large positive value that regulates the priority of constraints over the objective function. Setting a sufficiently large value for μ turns the problem into a constraint satisfaction problem

$$\mathcal{L}(x, \mu) \approx \sum_{i=1}^{i_{max}} \tilde{g}_i(x) + \sum_{j=1}^{j_{max}} \tilde{h}_j(x), \tag{4.24}$$

ignoring main objective function minimization in lieu of satisfying constraints.

4.4.1. Modeling objective and constraint functions

The objective function $f(x)$ can include various terms relating to kinematic and dynamic behaviour of the kinematic chain. An important thing to keep in mind is that these constraints

Minimizing state change

In kinematic planning $f(x)$ is often employed to keep the chain in a configuration close to the seed configuration x_{seed}

$$f(x) = (x - x_{seed}) W (x - x_{seed})^T . \tag{4.25}$$

This results in more predictable solutions to the problem, often avoiding non-modelled dynamic constraints resulting from large changes in configuration.

Pose control

We regulate the execution of a pose goal r using equality constraint functions of the type

$$\Delta r = 0 . \tag{4.26}$$

Convexifying (4.26) results in the familiar linear equation

$$h(x) = |J(x - x_k) - \Delta r| = 0 . \tag{4.27}$$

In the case of a Cartesian task-space, the error between target pose and current pose in (4.27) can be represented in the form of a 6d vector

$$\Delta r = \log\left(T_{target}^{-1}(a)\, T_{current}(x)\right) \tag{4.28}$$

The $\log$ operator here is the matrix $\log$ operator (further explain later). This vector can also be obtained using identities (3.18).

Obstacle avoidance

Obstacle avoidance can be handled as an inequality constraint in the NL optimization problem. The simplest way of implementing this is using the constraint function

$$g\left(x\right) = \left|p_{base} - p_{obstacle}\right| \leq R \tag{4.29}$$

This can in turn be extended to any number of obstacles. Collision checking libraries are often used to find a set of closest points for which to optimize.

Manipulability maximization

Having a metric M that describes the distance from singularity of the system Jacobian, we could use the scheme in (4.19) to optimize task execution. The manipulability metric of a kinematic chain has a couple of different definitions, the one used here is

$$M = \sqrt{\det\left(JJ^T\right)} \tag{4.30}$$

Here J is the kinematic chains Jacobian with respect to the Cartesian (although not necessarily) task-space. Through matrix manipulation it is possible to obtain a closed-form expression for the manipulability Jacobian

$$\frac{\partial M}{\partial x} = \frac{1}{2\sqrt{\det\left(JJ^T\right)}} Tr\left(\left(JJ^T\right)^{-1} \frac{\partial\left(JJ^T\right)}{\partial x}\right) \tag{4.31}$$

It might be possible to get this expression in symbolic form, but can be inefficient for real-time applications. Even though maximizing the manipulability measures increases the product of singular values, it does not mean that the smallest singular value will be sufficiently large. Another metric which minimizes size difference between singular values is the *condition number* κ

$$\kappa = \frac{\sigma}{\sigma_0}\,. \tag{4.32}$$

This value will ideally be close to 1, which means all the singular values are roughly the same magnitude. However this does not maximize the magnitude of singular values itself. The objective function to maximize a combination of these two metrics

$$f\left(x\right) = \sqrt{\frac{\kappa}{M}} \tag{4.33}$$

In the context of the mobile manipulator following both an end effector and base trajectory, this means that the optimization will utilize the mobile base orientation θ to maximize manipulability while ensuring that all the singular values are roughly the same magnitude.

4.4.2. Motion planning scheme

The full motion planning scheme can now be presented as an non-linear optimization problem. The parameter vector is in the case the configuration vector of the mobile manipulator as defined in (3.6)

$$\begin{aligned} \min_{q} \quad & \sqrt{\tfrac{\kappa}{M}} \\ \text{s.t.} \quad & \Delta r_{EE} = 0, \\ & (q_x - x_{goal})^2 + (q_y - y_{goal})^2 \leq R^2, \\ & q_{arm} \in [q_{min}, q_{max}] \,, \\ & q_\theta \in [-\theta, +\theta] \ . \end{aligned} \tag{4.34}$$

The objective function is the modified manipulability metric from (4.33), minimizing this function drives the manipulator away from singular configuration. The end-effector pose trajectory is introduced as a convex constraint function, alongside with the base position and orientation tolerances used for optimization. Additionally, we limit the search space to within the joint limits $[q_{min}, q_{max}]$.

4.4.3. The SQP algorithm

Sequential convex optimization optimization can be tackled using sequential quadratic programming (SQP) (Schulman et al., 2013; Xu et al., 2010), a powerful algorithm which repeatedly constructs and minimizes a convex approximation of the problem $\tilde{P}$ until constraints are satisfied.

Convexify For each point in the desired trajectory and the penalty value μ, we form convex cost functions at the operating point.

Minimize The convexified problems are minimized, where the parameter values being considered are subject to the trust region δ. This prevents the minimization process from taking big steps when the convex approximations do not reflect the real situation.

Trust Region At each iteration of the innermost for loop, a trust region δ for the QP problem is updated depending on how well the change proposed by the minimization algorithm reflect the change of the non-convex models of the cost functions.

Algorithm 2: The SQP algorithm

Data: $q = q_{init}$, f, g, h

Result: Configuration q satisfying f, g, h

for *PenaltyIteration = 1,2, ...* **do**
  for *ConvexifyIteration = 1,2, ...* **do**
    $\tilde{f}, \tilde{g}, \tilde{h} \leftarrow$ Convexify(f, g, h); `/* convexify problem */`
    for *TrustRegionIteration = 1,2, ...* **do**
      $\min_{q} \; G(\tilde{f}, \tilde{g}, \tilde{h})$; `/* minimize */`
      s.t. $q \in [q - \delta, q + \delta]$
      if *TrueImprove / ModelImprove* $\geq c$ **then**
        $\delta \leftarrow \tau^{+}\delta$; `/* expand trust region and proceed */`
        break;
      end
      $\delta \leftarrow \tau^{-}\delta$; `/* shrink trust region and repeat */`
    end
    if *Converged()* **then**
      break;
    end
  end
  if $SatisfiesConstraints$ **then**
    return q;
  else
    $\mu = k * \mu$;
  end
end

5. Software architecture

With the theoretical framework for state estimation and motion planning established in chapters 3 and 4, this chapter describes the software implementation of these ideas. Developing code that is intuitively structured is key in a research context, where it is often maintained and expanded by multiple parties. Maintaining a heavily tested core code-base which allows modular drop-in components for high level functions is an optimal development strategy for such cases. Such a structure expedites testing and data gathering procedures, as it enables quick start-up, user safety and damage prevention.

In section 5.1 development considerations are explained in more detail and a high-level overview of the software architecture is presented. Next, in 5.2 and 5.3 state estimation and motion planning pipelines are presented in more detail.Lastly, section 5.4 describes the control pipeline that relays user commands to the API and internal control loops.

5.1. Overview

Developing software for a complex system such as the Thing mobile manipulator is a time-consuming process, as each change requires extensive testing before it can be applied safely. Aside from maintaining a strict version control strategy with tools such as *git*, the developer needs to have an idea of what interaction a given change can have with the rest of the code-base as it is often inefficient to test every change in detail. Structuring the code in a way that decouples most of the heavily-tested core functionalities from specific user tasks is of great importance since it helps the developer maintain an idea of what certain changes can affect.

The testing procedure on the real robot suffers from a high set-up cost, which can be reduced by having start-up procedures stored in the form of ROS launch files. Having a simulation that reasonably models the robots behaviour also expedites testing, as some concepts remain largely unaffected by dynamic effects occurring in reality. Furthermore, platform-agnostic code is maintained, meaning that the software does not differentiate between the simulated and real systems, drastically reducing implementation time.

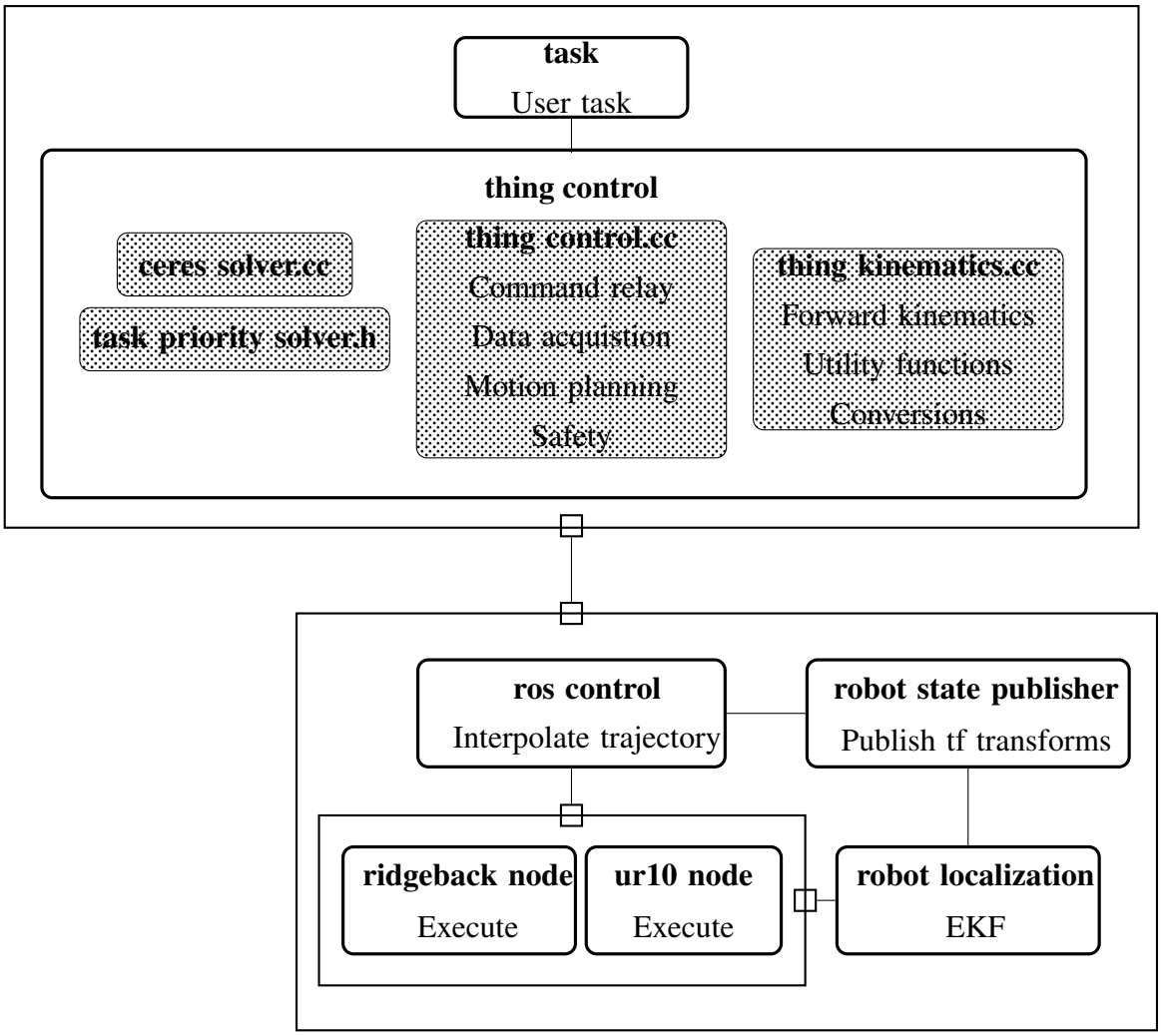


Figure 5.1: Overview of the software architecture. White rounded rectangles represent ROS packages, while cross-hatched rectangles represent individual libraries/headers/executables.

5.1.1. Architecture

The network-based nature of ROS systems allows a distributed software architecture. This enables keeping real-time dependent functionalities on the robot while running high level algorithms on a separate machine. Specifically, the control algorithms implemented in `ros_control` interface with the local UR10 and Ridgeback drivers and are ran on the robot's local machine which acts as a server. Task-specific programs are ran on a separate machine or client which connects to the Thing's ROS core via WiFi or ethernet.

Client

Figure 5.1 shows how the `thing_control` package forms a robust core of the code-base. It acts as a central node which processes trajectories and commands sent by the user, which includes:

- Receiving and processing data streams from the robot
- Sending joint-level commands to the lower-level control architecture
- Processing (solving) Cartesian trajectories sent by the user
- Monitoring incoming data streams and implementing security procedures

Receiving and sending data is done through the ROS topic interface; subscribers fetch structured data from network locations called topics which are populated using publishers.

At compile time, the user can choose a motion planner developed specifically for the Thing mobile manipulator. These planners inherit from a core part of the code-base, the `thing_kinematics` library. This library contains the common mathematical framework for the motion planners as well as various useful kinematics functions:

- Forward kinematics and Jacobian functions
- Pose error calculations
- Angle representation conversions
- Various matrix operations

Keeping this part of code in library form allows for easy transfer of these useful functionalities to other packages. The motion planners (also referred to as *solvers*) represent different approaches to the problem of solving a given Cartesian trajectory. Both the methods presented in 4.3, 4.4 are implemented as motion planning libraries.

Server

The computer located on the robot itself or server maintains the control and estimation pipeline on the robot and takes care of publishing the current state and status to the network.

Direct communication with the base and manipulator is achieved through their respective APIs, but in order to connect these libraries to our networked software structure the `ros_control` package is used. It provides a valuable set of tools that abstract the notion of a robot interface and enable quick and easy integration of APIs into ROS-based system architectures.

The transforms representing all of the chains located in the robots URDF[1] file resulting from state estimation are published thorugh the `robot_state_publisher` package. The package publishes the transforms in the form of standardized `\tf` ROS messages. The package is launched on the robot's local machine since the standardized structure of the control pipeline relies on the transforms published by this package.

[1] *Unified Robot Description Format* - Standard file format used for semantic robot descriptions.

5.2. State estimation

The EKF algorithm described in 3.3 is available as a component of the `robot_localization` package (Moore and Stouch, 2014). This package fuses odometry estimates provided by various sources, ideally resulting in a higher-accuracy estimate. The estimates need to be published to a ROS topic in the form of a `nav_msgs/odometry` message. Each state estimation node in `robot_localization` begins estimating the vehicle's state as soon as it receives a single measurement. If there is a holiday in the sensor data (i.e., a long period in which no data is received), the filter will continue to estimate the robot's state via an internal motion model.

5.3. Motion planning

Two motion planning algorithm have been developed, one implementing the *task-priority* (4.3) scheme and the other implementing the scheme in 4.34 cia the SQP (4.4) algorithm.

Unifying the implementations is a common linear algebra framework, the Eigen3 library[2]. The Eigen3 library is a powerful linear algebra toolbox capable of handling a vast array of dense and sparse matrix calculations. It is also templated and heavily optimized for reducing redundant value copying and memory management, making it the basis of many algorithm implementations across a range of fields.

5.3.1. Task-priority

Unlike the SQP implementations, task-priority is implemented using only the Eigen3 and C++ standard libraries. It relies on *singular value filtering* (4.12) and the *selectively damped least squares* (4.11) formulation to maintain numeric stability in the pseudoinverse operation. These functions have been developed from scratch in highly abstracted template form for the purposes of this work and are publicly available[3].

5.3.2. Sequential convex optimization

Implementation of the sequential convex optimization algorithm is done using Google Ceres solver as the underlying convex optimization subprogram. The objective and convexified constraint functions are implemented as classes in order to be compatible with the solver. Ceres is configured to run using the Levenberg-Marquardt trust-region strategy with the Broyden–Fletcher–Goldfarb–Shanno algorithm being used for performing line search.

[2] http://eigen.tuxfamily.org/index.php?title=Main_ Page

[3] because of it's compact form it was also included in the appendix

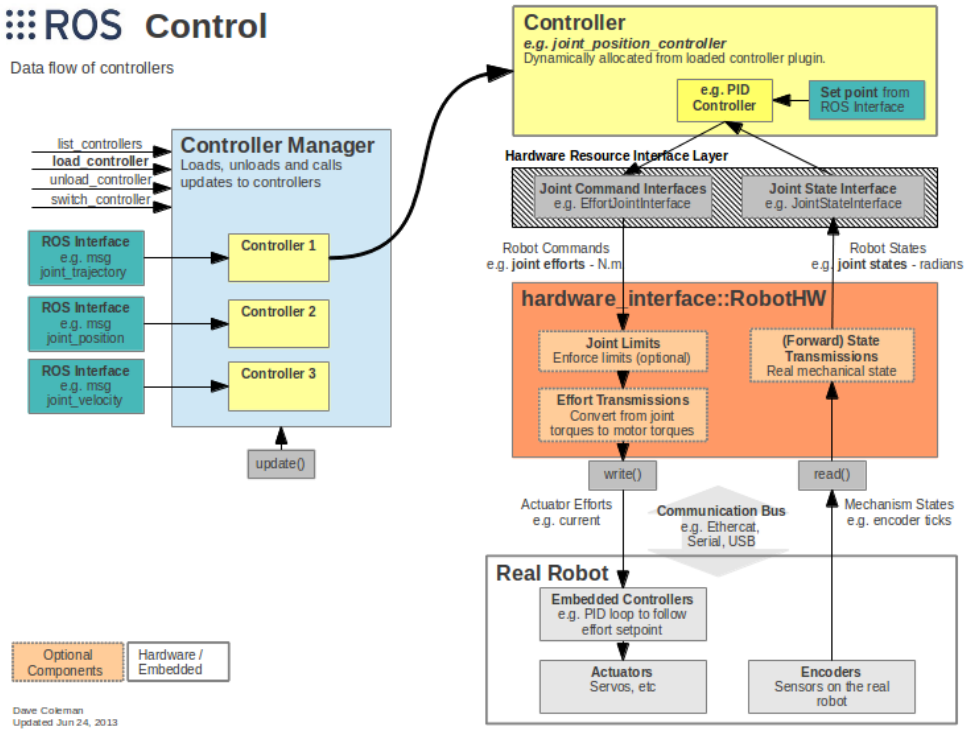


Figure 5.2: Scheme of ros_ control functionalities

5.4. Control

The underlying control architecture of the Thing mobile manipulator relies heavily on the internal APIs of both the UR10 and Ridgeback. Thus, the main challenge in implementing the software architecture lies integration of control inputs and outputs rather than parametrization of a control loop. The `ros_control` package functions as an integration tool for robot APIs into the networked structure of ROS-based systems. The `hardware_interface` **??** class is used to wrap the read and write methods of the respective APIs in to function handles. These handles can then be used by a selection of controllers within the package to either forward commands or implement a control loop. A ros controller has many types and is configured using a yaml file. Depending on the interface, this controller will forward commands or actually control the signal using tunable PID parameters. Additionally, we use the joint trajectory controllers which can handle and interpolate multiple timed trajectory points.

6. Results

6.1. Simulation

6.2. Real

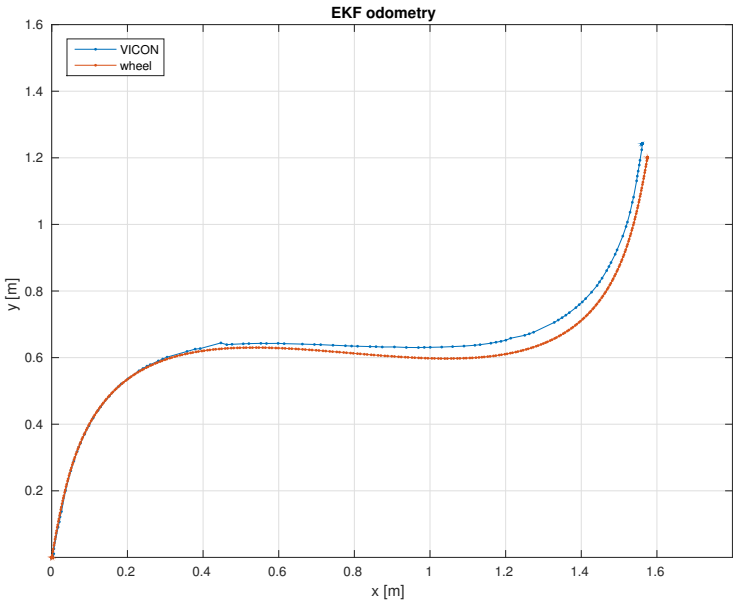


Figure 6.1: Simulated localization

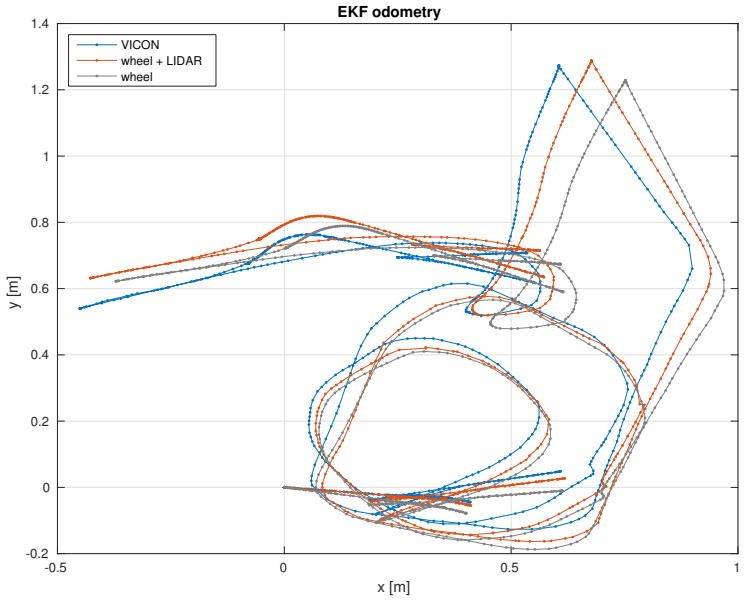


Figure 6.2: Simulated localization

7. Conclusion

Conclusion.

BIBLIOGRAPHY

John Baillieul. Kinematic programming alternatives for redundant manipulators. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 722–728. IEEE, 1985.

Anthony Bloch, John Baillieul, Peter Crouch, Jerrold E Marsden, Dmitry Zenkov, Perinkulam Sambamurthy Krishnaprasad, and Richard M Murray. *Nonholonomic mechanics and control*, volume 24. Springer, 2003.

Samuel R Buss and Jin-Su Kim. Selectively damped least squares for inverse kinematics. *Journal of Graphics tools*, 10(3):37–49, 2005.

Stephen K Chan and Peter D Lawrence. General inverse kinematics with the error damped pseudoinverse. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 834–839. IEEE, 1988.

Stefano Chiaverini. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Transactions on Robotics and Automation*, 13 (3):398–410, 1997.

Stefano Chiaverini, Bruno Siciliano, and Olav Egeland. Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator. *IEEE Transactions on control systems technology*, 2(2):123–134, 1994.

Adria Colomé and Carme Torras. Redundant inverse kinematics: Experimental comparative review and two enhancements. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5333–5340. IEEE, 2012.

Adria Colomé and Carme Torras. Closed-loop inverse kinematics for redundant robots: Comparative assessment and two enhancements. *IEEE/ASME Transactions on Mechatronics*, 20(2):944–955, 2015.

Moharam Habibnejad Korayem and H Ghariblu. Analysis of wheeled mobile flexible manipulator dynamic motions with maximum load carrying capacities. *Robotics and Autonomous Systems*, 48(2):63–76, 2004.

Moharam Habibnejad Korayem, HN Rahimi, and A Nikoobin. Mathematical modeling and trajectory planning of mobile manipulators with flexible links and joints. *Applied Mathematical Modelling*, 36(7):3229–3244, 2012.

LA Leshin, PR Mahaffy, CR Webster, Michel Cabane, Patrice Coll, PG Conrad, PD Archer, SK Atreya, AE Brunner, A Buch, et al. Volatile, isotope, and organic analysis of martian fines with the mars curiosity rover. *Science*, 341(6153):1238937, 2013.

Signe Moe, Gianluca Antonelli, Andrew R Teel, Kristin Y Pettersen, and Johannes Schrimpf. Set-based tasks within the singularity-robust multiple task-priority inverse kinematics framework: General formulation, stability analysis, and experimental results. *Frontiers in Robotics and AI*, 3:16, 2016.

T. Moore and D. Stouch. A generalized extended kalman filter implementation for the robot operating system. In *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014.

Yoshihiko Nakamura. *Advanced robotics: redundancy and optimization*. Addison-Wesley Longman Publishing Co., Inc., 1990.

Yoshihiko Nakamura, Hideo Hanafusa, and Tsuneo Yoshikawa. Task-priority based redundancy control of robot manipulators. *The International Journal of Robotics Research*, 6 (2):3–15, 1987.

Evangelos Papadopoulos and Yves Gonthier. A framework for large-force task planning of mobile and redundant manipulators. *Journal of Robotic Systems*, 16(3):151–162, 1999.

John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, pages 1–10, 2013.

Siciliano B Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. In *proceeding of 5th International Conference on Advanced Robotics*, volume 2, pages 1211–1216, 1991.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic robotics. 2005. *CURRICULUM VITAE*, 2005.

Jianxin Xu, Wei Wang, and Yuanguang Sun. Two optimization algorithms for solving robotics inverse kinematics with redundancy. *Journal of Control Theory and Applications*, 8(2):166–175, 2010.

Yoshio Yamamoto and Xiaoping Yun. Coordinating locomotion and manipulation of a mobile manipulator. In *Decision and Control, 1992., Proceedings of the 31st IEEE Conference on*, pages 2643–2648. IEEE, 1992.

Appendix A
DH parameters

Link	d_i	θ_i	a_i	α_i
1	0	$\frac{\pi}{2}$	0	$\frac{\pi}{2}$
2	q_x	$\frac{\pi}{2}$	0	$\frac{\pi}{2}$
3	q_y	$\frac{\pi}{2}$	0	$\frac{\pi}{2}$
4	0	q_θ	0	0
5	0.653	0	0.27	0
6	0	$\frac{\pi}{2}$	0.01	0
7	0.1273	q_0	0	$\frac{\pi}{2}$
8	0	q_1	-0.612	0
9	0	q_2	-0.5723	0
10	0.163941	q_3	0	$\frac{\pi}{2}$
11	0.1157	q_4	0	$-\frac{\pi}{2}$
12	0.0922	q_5	0	0

(A.1)

Table A.1: Thing's DH parameters

Appendix B
Robot operating system

ROS is an open source software development tool for implementing robotics software. It provides the opportunity of hardware abstraction, low level device control, implementation of commonly used functionalities, messages between different processes and package management. It provide tools and libraries which utilize the the opportunity of communicating between disturbed computers, obtaining, writing and running code.

ROS has three different levels of concepts

- **The file system level**

  Handles the main unit for a ROS system which is packages. A package may include data sets, ROS dependent libraries, configure files etc. to define a ROS process. In ROS a process is denoted as a node.

- **The computation graph level**

  Handles the communication of the peer to peer network of the system in which data is processed. Through the computation graph level, the different nodes can communicate with each other by messages. When a node is sending data it is said to be publishing a topic. The different nodes can then subscribe to this topic to get the information that is published.

- **The Community level**

  ROS has a huge community which contain distribution of software installations, repositories and documentation of ROS. It also has a question and answer section with ROS related topics.
  This community makes the process of learning the system considerably easier.

From Global to Local: Maintaining Accurate Mobile Manipulator State Estimates Over Long Trajectories

Abstract

Abstract.

Keywords: Keywords.

Naslov

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.