Playing Tetherball with Compliant Robots

Hany Abdulsamad **Tom Buchholz** **Tobias Croon** **Mario El Khoury**
Institute for Intelligent Autonomous Systems
Technische Universität Darmstadt

Abstract

In this paper we present a setup of two BioRob robotic arms playing the game of tetherball. We start by introducing the robotic arms and the communication framework we have established around them. We go on to explain our vision approach of detecting the ball's position. In preparation for the main task we create an automated vision calibration task to be able to calculate the needed transformations between all setup components. As a first step towards hitting the ball we use a simplified approach of modeling the ball's movement with the equations of a spherical pendulum and solve standard robotic problems like constrained inverse kinematics and joint-trajectory planning. For the much more complex model of a tetherball we derive the equations using the Lagrange-d'Alembert Principle and compared the performance of two state estimators for hidden model states. Finally we combine all submodules into one task that implements the tetherball game.

1 Introduction

Tetherball is a two player game where a ball hanging from a pole with a string is hit repeatedly between the participants. The goal is to wind the string around the pole as far as possible in one direction without giving the opponent the chance to unwind it. As related work we mention [1] where the authors present a new platform for playing table tennis with a Barrett robotic arm. The platform is the basis for extensiv research in learning complex skills by generalizing a set of motor primitives [2] [3] [4]. In [5] a simple tetherball setup is presented and used in learning and choosing from multiple solution of a motor skill task. The aim of our work is to build a new platform for future research projects and applications. In our setup the players are represented by two elastic robots imitating a human arm. For vision we use a Microsoft Kinect sensor.

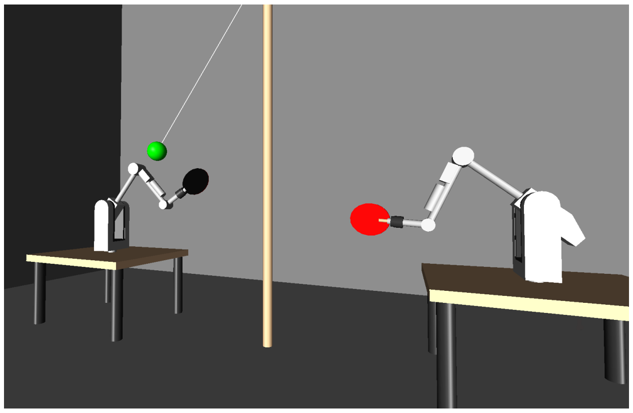


Figure 1: BioRob Tetherball Setup: The vision sensor is placed above the robots for optimal coverage.

2 Physical Setup

Two robotic arms are fixed on bases that stand 2 meters apart. Between the robots we have a 3 meter pole with a colored ball hanging down from a 2 meter string. For hitting the ball we mount table tennis paddles as end effectors. The Kinect sensor delivers color and depth images at a rate of 30Hz and is mounted on the ceiling for optimal viewing angle. In Fig. 1 we present an illustration of our tetherball setup in a simulation environment.

2.1 The BioRob

The BioRob is a lightweight elastic robotic arm with 6 DoF. The elasticity is introduced with springs in 4 out of the 6 joints. This improves compliance and safety, but also makes a joint-side control of the robot harder. The combination of elastic joints and an efficient distribution of mass allows the robot to reach high joint speeds. The kinematic chain of the BioRob is described based on the Denavit-Hartenberg convention. The D-H parameters describe the spatial transformation of the frame of reference of each joint relative to the previous one starting from the first joint as described in [7]. Table 1 lists the values of the four D-H parameters for each joint.

Table 1: D-H Parameters of the BioRob

i	θ	d	a	α
1	q_1	0.284	0	$\pi/2$
2	$q_2+\pi/2$	0	0.305	0
3	$q_3+\pi/2$	0	0	$\pi/2$
4	$q_4+\pi$	0.305	0	$\pi/2$
5	$q_5+\pi$	0	0	$\pi/2$
6	$q_6+\pi$	0.220	0	$\pi/2$

The transformations denoted by the parameters are to be executed in the current frame of reference and in the same order in which they are listed. The parameter θ denotes the rotation around the Z-axis, while d and a stand for the translation along the Z- and X-axis respectively and α is the rotation around the X-axis. The static joint angle offsets are specified by q^*.

The kinematic redundancy of having 6 DoF makes the BioRob suitable for the tetherball task, because it enables the user to specify not only the position of the end effector in the cartesian space but also the orientation of the paddle which is critical when trying to hit the ball. A schematic of the full kinematic structure is shown in Fig. 2.

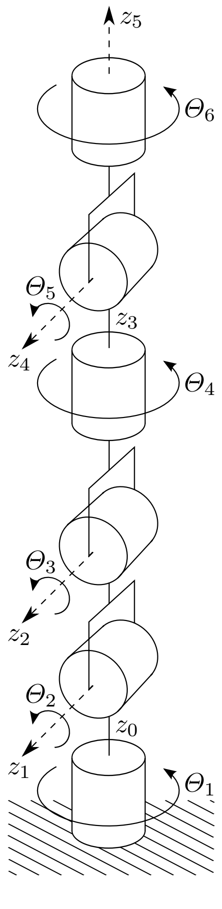


Figure 2: Kinematic Chain of the BioRob

3 Communication Structure

The control station consists of two PCs running under Linux. The primary PC is running a user interface and is connected directly to one BioRob and to the Kinect. The secondary PC has no UI and controls the second BioRob. The whole structure is depicted in Fig. 3.

3.1 Simulation Lab (SL)

We use the OpenGL based physical simulation environment SL [12] as a user interface on the primary PC. The software runs several processes including a vision servo where the position of the robot is depicted graphically and the task servo where user specific tasks are initialized and executed. The simulation communicates with the robot over a shared memory protocol from which SL is able to read the robot's current joint positions $\mathbf{q}$ and velocities $\dot{\mathbf{q}}$ as well as the position of the ball $\mathbf{p} = \{x, y, z\}$ and to write the calculated feed-forward gravity compensation and PD-controller feedback action vector $\mathbf{u}$ for the robot to execute.

Because SL is only able to simulate one robot at a time, we defined the whole two BioRob setup as one kinematic chain, where the two robotic arms are connected at the base with a static joint and each end of the chain has its separate end effector. This leads to the expansion of all relevant joint and action vectors to thirteen dimensions. The first six of them belong to the first robot and the last six to the second one. The seventh element represents the constrained connecting joint which cannot be manipulated.

3.2 Kinect Data Processing

The Kinect sensor is used in vision calibration in addition to its main objective of tracking the green ball. A separate module on the primary PC, running outside SL, initializes and starts the sensor. This module, written in C++, utilizes the OpenCV library [13] to read out the RGB and depth images of the sensor and to extract the ball's coordinates and write them into shared memory for SL to use them.

3.3 PC-PC-BioRob Interaction

The communication with the BioRob takes place over a dedicated Ethernet-Card. The underlying layer of the connection is provided through the software framework ROS [18]. We programmed two custom ROS modules to control the robot on two levels. The first module is the SL-BioRob-Interaction-Module which allows the user to send several commands from SL directly to the robot like choosing the desired control loop and enabling the robot's actuation. The second module is the SL-Controller-Module which manages process of reading and writing the joint angles, veloc-

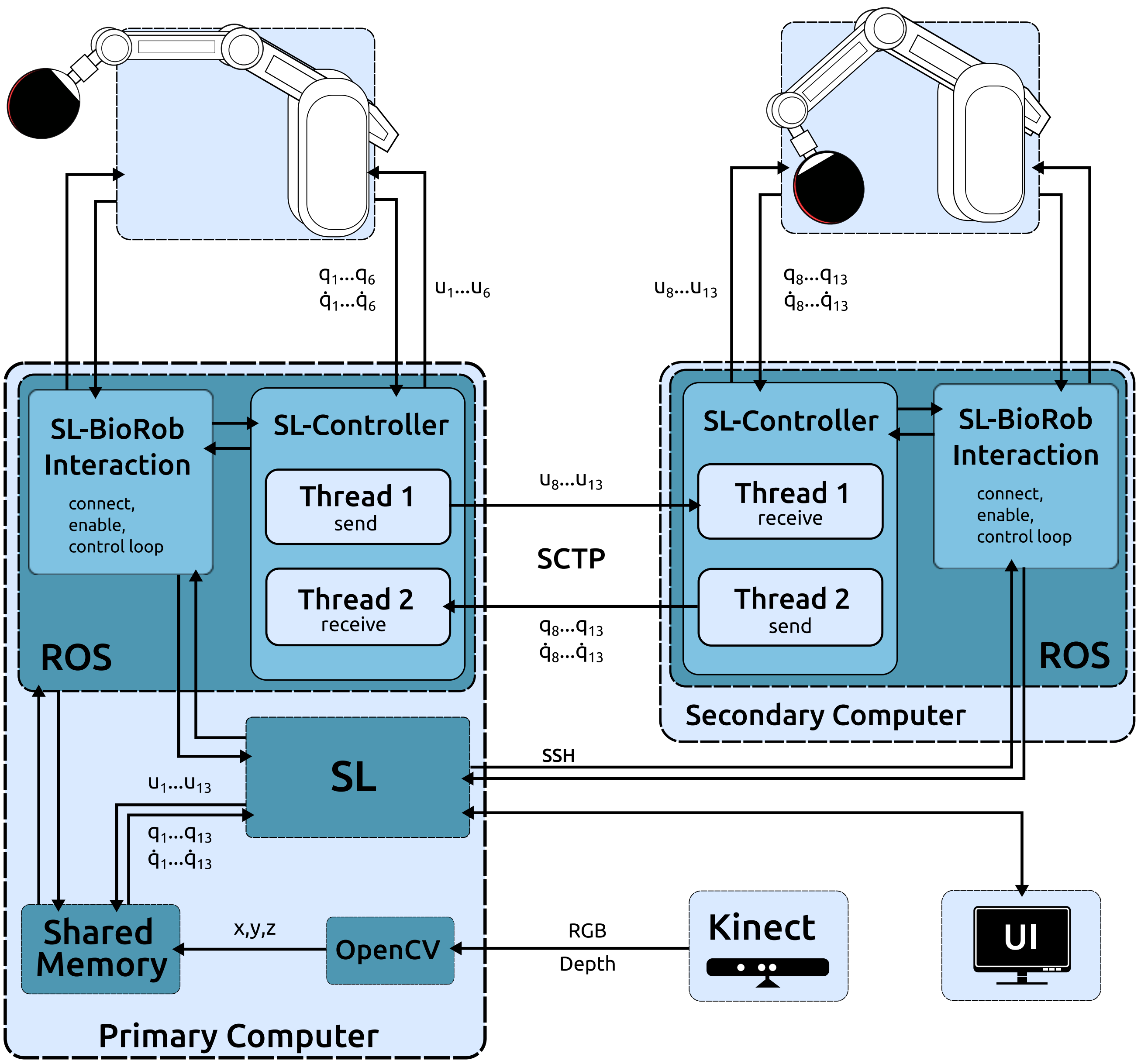


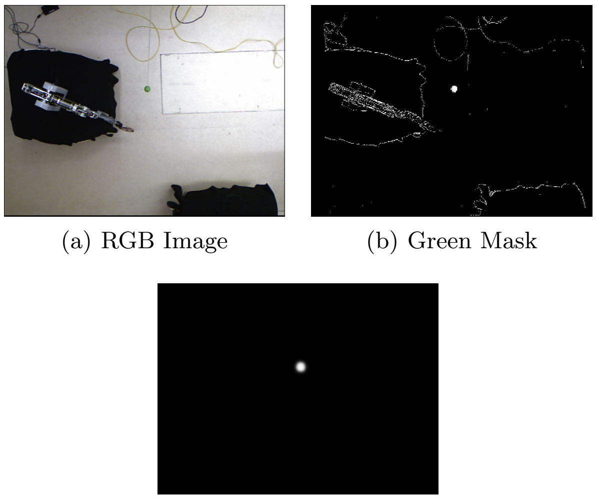
Figure 3: Tetherball Communication Schematic

ities and actions between SL and the BioRob. In the case of the BioRob connected to the primary PC, this exchange is done directly over shared memory. For the second BioRob however we needed to establish a local network connection between the two controller modules. For this connection we used the Stream Control Transmission Protocol (SCTP). We choose to use SCTP because it combines multiple advantages from both TCP and UDP. The most important are preservation of message boundaries, reliable data transfer and ordered data delivery [19].
We divided the sending and receiving on both sides into multiple real time threads from the Orocos Real-Time Toolkit [17]. The threading technique is important in order to sustain a continuous non-blocking connection to the robot at a rate of 1kHz. As different threads try to read and manipulate the same data structures at the same time, we regulate the access rights of each thread with the help of mutexes.

4 Ball Detection

The Kinect sensor obtains color and depth images via the OpenNI library. With the help of the OpenCV library it is possible to detect the ball as a circle and determine the cartesian position of its center.

4.1 RGB Image Based Ball Detection



(a) RGB Image (b) Green Mask

(c) Detected Circle

Figure 4: RGB Based Ball Detection

The existing approach [6] relied on recognizing the ball as a green circle in the RGB image. This is done by using a color specific mask that discards (blackens) non relevant pixels, Fig. 4(b). The resulting image is then filtered by eroding noisy pixels and leaving the ball as the only concentration of green pixels to be considered. This cloud of pixels is then dilated and blurred as a preparation for using the Hough Transform to detect the center, Fig. 4(c). The Hough Transform delivers the position of the ball in the plane. A depth value is then assigned by masking the depth image with the area defined by the circle's diameter and averaging over the depth values of all pixels inside the circle.

The problem with this method lies in the time discrepancy between the RGB and depth image. This means that the depth information belonging to the area around the detected circle in the RGB image is not up to date and might even not contain any depth values of the ball, only those of the ground or the surroundings objects. A second problem arises because of a trade off between detecting circles as often as possible and the quality of the detected circles. This leads to imperfect circles that might go over the masked green areas. Averaging over the depth values of those false pixels will corrupt the depth measurement of the ball. In Fig. 5 we illustrate the problem on a sample recording from the Kinect, where we can clearly see that the RGB image and the depth image are not synchronized.

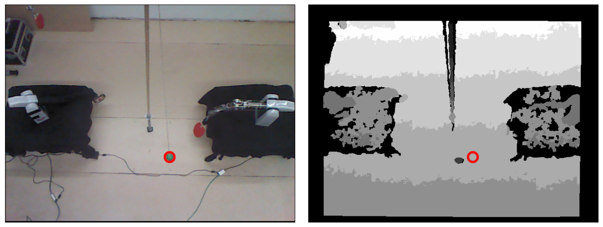
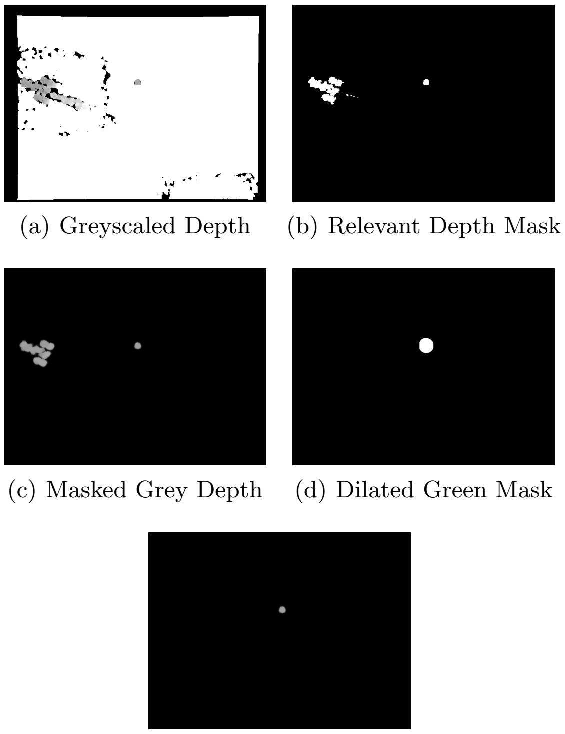


Figure 5: Misalignment of RGB and Depth

4.2 Depth Image Based Ball Detection

In order to overcome the previously mentioned problems in the depth measurement we introduce a different algorithm to determine the ball's position. Although the new approach is also based on the Hough Transform, the circle detection takes place in the depth image instead of the RGB.
We start by grayscaling the depth image in the relevant depth band as inferred from ball movement, Fig. 6(c). The grayscaling is important when using the Hough Transform in its Hough Gradient variation. We then scan the RGB image for green pixels, Fig. 6(d), in order to isolate the green ball, and apply the result, after noise erosion and dilation, as a mask to the grayscaled depth image. After some Gaussian blurring to improve the performance of the circle detection, we apply the Hough Transform to the green masked grayscaled depth image. The center of the circle and the average over all depth values in the circle deliver the final ball position. In Fig. 7 we show the important steps of this method.


(a) Greyscaled Depth (b) Relevant Depth Mask

(c) Masked Grey Depth (d) Dilated Green Mask

(e) Green Masked Depth

Figure 6: Depth Based Ball Detection

The main advantage of this approach is that we are able to avoid the problem with the misaligned RGB and depth. All components of the ball's position are inferred from the depth image, while the RGB image does not play a critical role like in the previous algorithm and serves only as color mask, in whose vicinity we search for the depth circle to limit the number of false detections, Fig. 7.
A comparison between the performance of the two detection algorithms is shown in Fig. 16.

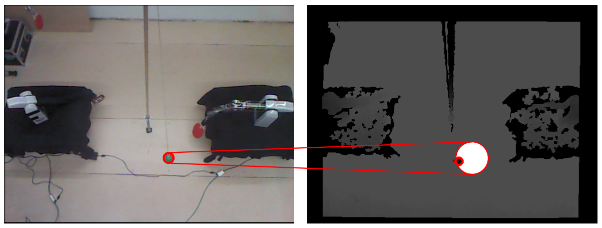


Figure 7: 3D Position from the Depth Image

5 Vision Calibration Task

Because we need the position of the ball relative to different frames of reference (FoR), we have to determine the transformation matrices between them. The ball is first detected in the Kinect's frame of reference. For the estimation of the ball's trajectory we have to transform the Kinect data to the pivot's frame of reference which then also has to be transformed into the frame of reference of the BioRob in order to hit the ball. For this reason we have created a fully automated vision calibration task which calculates all the needed transformations under different settings.
The task consists of three stages. The first two determine the transformation matrix ${}^{p}\mathbf{T}_{c}$ [*][†]between the Kinect's frame of reference and the pivot while the third stage determines the transformation ${}^{r}\mathbf{T}_{c}$ between the Kinect and each of the BioRobs. Those two matrices also yield the transformation matrix between the pivot and the two BioRobs respectively:

$$ {}^{r}\mathbf{T}_{p} = {}^{r}\mathbf{T}_{c} \cdot {}^{p}\mathbf{T}_{c}^{-1} \tag{1}$$

Here we use the homogenous coordinate representation to combine the rotation matrix and the translation vector into one transformation matrix $\mathbf{T}$ of the form:

$$\mathbf{T} = \left[\begin{array}{ccc|c} & & & t_x \\ & \mathbf{R} & & t_y \\ & & & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2}$$

5.1 Kinect-Pivot Transformation

In determining the transformation matrix ${}^{p}\mathbf{T}_{c}$ we implemented two stages. First we calculate the position of the pivot as a point in the Kinect's frame of reference. Then we measure the ball's resting position, also in the Kinect's frame of reference. Those two points give us a direction of the Z-axis of the pivot's frame of reference as from the Kinect's point of view and this allows us to infer the transformation matrix. In the following sections we go into the details of both stages.

5.1.1 Pivot Point in the Kinect Frame

In this stage we let the ball move around the Kinect's field of vision without the pole, so that ideally the movement covers a large area. During the movement we collect all the occurring positions of the ball in a matrix $\mathbf{M} = \begin{bmatrix}\mathbf{x}_b^c & \mathbf{y}_b^c & \mathbf{z}_b^c\end{bmatrix}$[‡]. Assuming the string always stays taut, all collected points lie on a sphere with the pivot point $\mathbf{p}_p^c$ as its center and a radius equal to the length of the string. Using nonlinear least squares, while taking the length of the string into consideration, we can search for a sphere that fits the data and the radius. The objective function, Eq. 3, of this optimization problem minimizes the euclidian distance

[*]For matrices the lower right index denotes the current FoR while the upper left denotes the target FoR

[†]Notation: BioRob(r), Kinect(c), Pivot(p), Ball(b), End Effector(e)

[‡]For vectors the lower right index is unique to the vector while the upper right denotes the current FoR.

between the data and the sphere defined by the three coordinates of the pivot point representing the function parameters.

$$min\left\{\sum_{i=1}^{N}\left(l-\sqrt{(x_p^c-\mathbf{x}_b^c)^2+(y_p^c-\mathbf{y}_b^c)^2+\ldots+(z_p^c-\mathbf{z}_b^c)^2}\right)^2\right\} \tag{3}$$

To solve this minimization problem we implement the Levenberg-Marquardt algorithm (see [8]) as provided by the LEVMAR open source library [14].

5.1.2 Resting Position Relative to Kinect

In this stage we just let the ball hang without moving, while we collect a number of points. The ball's resting position $\mathbf{p}_{b,0}^c$ is then calculated as the mean of the collected points to compensate for noise and outliers.

5.1.3 Kinect-Pivot-Matrix ${}^p\mathbf{T}_c$

From the two points that we already have, it is easy to define the Z-axis $\mathbf{w}_p^c$ of the pivot's frame of reference as seen from the Kinect:

$$\mathbf{w}_p^c=\frac{\mathbf{p}_{b,0}^c-\mathbf{p}_p^c}{\|\mathbf{p}_{b,0}^c-\mathbf{p}_p^c\|} \tag{4}$$

In order to define its X-axis $\mathbf{u}_p^c$, we just take the X-axis of the Kinect and adopt its direction in the X-Y-plane and correct the direction in X-Z-plane by rescaling the Z-coordinate so that the two axes $\mathbf{u}_p^c$ and $\mathbf{w}_p^c$ are perpendicular:

$$\begin{aligned}\tilde{\mathbf{u}}_p^c&=\begin{bmatrix}1 & 0 & -\frac{\mathbf{w}(1)_p^c}{\mathbf{w}(3)_p^c}\end{bmatrix}\\ \mathbf{u}_p^c&=\frac{\tilde{\mathbf{u}}_p^c}{\|\tilde{\mathbf{u}}_p^c\|}\end{aligned} \tag{5}$$

Notice that the only important attribute of the pivot's FoR is that its Z-Axis is pointing downward in a parallel line to the gravitational force. The rotation of the X-Y-plane around this axis is completely irrelevant to our aim. Using the cross product of the normalized Z- and X-axis we can now calculate the Y-axis $\mathbf{v}_p^c$:

$$\mathbf{v}_p^c=\mathbf{w}_p^c\times\mathbf{u}_p^c \tag{6}$$

With all axes of the pivot system now defined relative to the Kinect, we can easily choose four points whose position we already know in both frames and infer the transformation matrix. In our case we choose the origin of the pivot's FoR $\mathbf{p}_p$ which we have already calculated in addition to the three other known points at the tip of the unit vectors of the calculated axes:

$$\mathbf{X}_i^p=\begin{bmatrix}0&1&0&0\\0&0&1&0\\0&0&0&1\\1&1&1&1\end{bmatrix}$$

$$(\mathbf{X}_i^c)^T=\begin{bmatrix}(\mathbf{p}_p^c)^T & 1\\(\mathbf{p}_p^c+\mathbf{u}_p^c)^T & 1\\(\mathbf{p}_p^c+\mathbf{v}_p^c)^T & 1\\(\mathbf{p}_p^c+\mathbf{w}_p^c)^T & 1\end{bmatrix}$$

$${}^p\mathbf{T}_c=\mathbf{X}_i^p\cdot(\mathbf{X}_i^c)^{-1} \tag{7}$$

5.2 Kinect-BioRob Transformation

In the third step of the vision calibration task the transformation matrix ${}^r\mathbf{T}_c$ between the Kinect's frame of reference and each of the BioRobs is calculated. In order to estimate the transformation we move the BioRob's end effector around so it covers as much of the Kinect's field of vision as possible. While the robot is moving we collect a large number of samples n of the cartesian position of the BioRob's end effector in the Kinect's frame of reference. We do this by detecting the red side of paddle. At the same time, by using the forward kinematics of the robot, we collect samples of the end effector's cartesian position in the BioRob's frame of reference. This measurement results in two $4\times n$ matrices $\mathbf{X}_e^r$ and $\mathbf{X}_e^c$. Now the transformation matrix can be estimated using linear least squares and the Moore-Penrose pseudoinverse $(\mathbf{X}_e^c)^+$:

$${}^r\mathbf{T}_c=\mathbf{X}_e^r\cdot(\mathbf{X}_e^c)^+ \tag{8}$$

For simulation in SL, we also need the distance and the angles between the two BioRobs. These can easily be calculated. The resulting matrix contains the translation vector and the Euler-angles:

$${}^{r1}\mathbf{T}_{r2}={}^{r1}\mathbf{T}_c\cdot{}^c\mathbf{T}_{r2} \tag{9}$$

6 Pendulum Task

Before trying to implement the actual tetherball task, we try a simpler setup in which the pole is left out. This new setup is similar to that of the tetherball game but has a simpler mechanical system. This task serves as a backbone in which we test and implement several subtasks that we can later employ in the tetherball task. The main difference between the two setups lies in the movement of the ball. The pendulum task implements a simpler model in which, on the contrary to tetherball, the length of the string is always constant and the pivot point is stationary.

We start be deriving the model equations and then go over the subtasks like filtering and estimation, inverse kinematics and joint-trajectory planning.

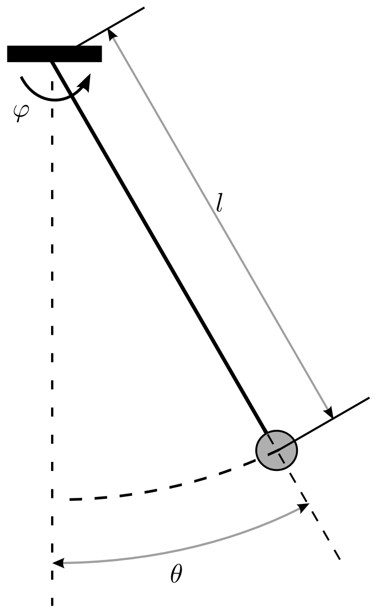


Figure 8: Spherical Pendulum

6.1 Spherical Pendulum Model

In Fig. 8 we present a simplified schematic of a spherical pendulum. The position of the ball at the end of the string (point mass) can be described in different coordinate systems. However because the system has multiple rotational symmetries, it is best described in the spherical coordinates. A direct benefit of this representation is that it breaks down into only two generalized coordinates[§], because the string l is expected to be always taut at all times. The two generalized coordinates are thus the azimuth angle φ and the polar angle θ. Another reason to go for the spherical coordinates is that the cartesian representation would result in a set of differential algebraic equations which are harder to handle.
We derive the equations of motion with the Lagrangian Formalism [10], which states that the Lagrangian energy function $L = T - U$, with T as the kinetic energy and U as the potential energy, must satisfy the Euler-Lagrange equation of the form:

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{q}_i}) - \frac{\partial L}{\partial q_i} = 0 \tag{10}$$

with $q_i, \dot{q}_i$ symbolizing the generalized coordinates $\{\varphi, \theta\}$ and their time derivatives $\{\dot{\varphi}, \dot{\theta}\}$.

[§]Generalized coordinates in mechanics are the minimal combination of system states that fully define the position. In other words they stand for the system's DoF

Because it is more comprehensible, we begin by writing down both energy functions in the cartesian space:

$$T = \frac{m}{2}(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) \tag{11}$$

$$U = -mgz \tag{12}$$

m, g here stand for the mass and the gravitational acceleration respectively. Now to map the energies to the spherical space we apply the following transformation for the states:

$$\begin{aligned} x &= l\sin(\theta)\cos(\varphi) \\ y &= l\sin(\theta)\sin(\varphi) \\ z &= l\cos(\theta) \end{aligned} \tag{13}$$

and we compute the state derivatives:

$$\begin{aligned} \dot{x} &= l\dot{\theta}\cos(\theta)cos(\varphi) - l\dot{\varphi}\sin(\theta)\sin(\varphi) \\ \dot{y} &= l\dot{\theta}\cos(\theta)\sin(\varphi) + l\dot{\varphi}\sin(\theta)\cos(\varphi) \\ \dot{z} &= -l\dot{\theta}\sin(\theta) \end{aligned} \tag{14}$$

The resulting Lagrangian L is then given by:

$$\begin{aligned} L &= T - U \\ &= \frac{m}{2}(l^2\dot{\theta}^2 + l^2\dot{\varphi}^2\sin^2(\theta)) + mgl\cos(\theta) \end{aligned} \tag{15}$$

By calculating the required derivatives of L and applying them to the Euler-Lagrangian equation, Eq. 10, we arrive at the following equations of motion:

$$\ddot{\theta} = \frac{l\dot{\varphi}^2\sin(\theta)\cos(\theta) - g\sin(\theta)}{l} \tag{16}$$

$$\ddot{\varphi} = \frac{-2\dot{\varphi}\dot{\theta}\cos(\theta)}{\sin(\theta)} \tag{17}$$

We transform these equations into their state space form:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{\varphi} \\ \ddot{\varphi} \end{bmatrix} = \begin{bmatrix} x_2 \\ x_4^2\sin(x_1)\cos(x_1) - \frac{g}{l}\sin(x_1) \\ x_4 \\ -\frac{2x_2x_4\cos(x_1)}{\sin(x_1)} \end{bmatrix} \tag{18}$$

where the resulting system of differential equations is of the fourth order. Since an analytical solution cannot be found we solve these equations numerically from four initial conditions. Because the Kinect provides only the cartesian position overlaid with noise, we transform each sample into spherical coordinates then differentiate and average the spherical velocities over a window of 5 samples. We solve this initial value problem with a Runge-Kutta solver of the fourth order and a dynamic step control as provided by the open source GSL library [15].

6.2 Vision Filtering

Although we were able to significantly improve the Kinect measurements by detecting the ball in the depth image, we still faced problems with accuracy especially when trying to determine the spherical velocities in order to be able to estimate the trajectory by integrating the initial conditions. For this reason, we have implemented a solution specific to the pendulum system by exploiting geometrical constraints.

6.2.1 Projection onto a Sphere

Assuming that the string is always under tension and that it is attached to a fixed point, the pivot $\mathbf{O}^p$, which is not identical to the origin of the Kinect $\mathbf{K}^p$, we know that the ball can only move on the surface of a sphere. The center of this sphere is the pivot $\mathbf{O}^p$ and its radius is the string's length l. Taking the model into consideration we are able to correct outliers in the depth measurement and improve the position of the ball in the X-Y-plane.
We assume that although the measurement of the ball $\mathbf{p}^p_m$ is faulty, it is still aligned with the actual position of the ball on the sphere $\mathbf{p}^p_s$ as shown in Fig. 9.

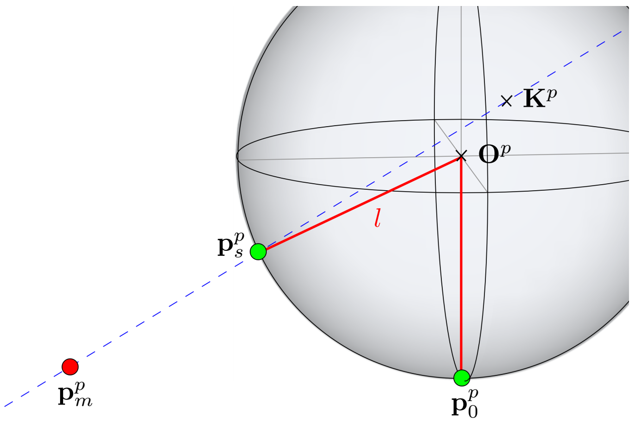


Figure 9: Projection of Ball Position onto a Sphere

This means that the correct ball position $\mathbf{p}^p_s$ lies somewhere on the line connecting the points $\mathbf{K}^p$ and $\mathbf{p}^p_m$ and satisfies both the sphere and the line equations which are described by:

$$(x^p_s - x^p_o)^2 + (y^p_s - y^p_o)^2 + (z^p_s - z^p_o)^2 = l^2 \tag{19}$$

$$\begin{bmatrix} x^p_s \\ y^p_s \\ z^p_s \end{bmatrix} = n \cdot \begin{bmatrix} x^p_m \\ y^p_m \\ z^p_m \end{bmatrix} \tag{20}$$

We solve these equations for $\mathbf{p}^p_s$ by calculating the lower intersection point of the sphere and the line. In Fig. 17 we show the improvement achieved by using this projection.

6.3 Inverse Kinematics

When the estimation of the future trajectory is done and a hitting point has been chosen this point has to be transformed from the cartesian to the joint space of the robot. This transformation is the inverse kinematics. In the beginning we chose to calculate it using the iterative Jacobian-Transpose method. Compared to the Inverse-Jacobian the Jacobian-Transpose offers important numerical stability when the rate of convergence is not an issue [9].

$$\dot{\mathbf{q}} = \gamma \mathbf{J}^{\mathbf{T}}(\mathbf{q})\dot{\mathbf{x}} \tag{21}$$

As we improved our implementation of the task we also chose to enhance our inverse kinematics algorithm to include joint limits as well as consider the orientation of the paddle at the chosen hitting point. To this end we formulate the inverse kinematics as a constrained optimization problem. The objective function of this problem has 6 parameters represented by the joint angles with 6 constraints. This function, Eq. 22, compares the current cartesian end effector position and orientation (Euler angles) as delivered by the forward kinematics function with the goal position and orientation:

$$\{x, y, z, \alpha, \beta, \gamma\} = \text{ForKin}(\mathbf{q})$$

$$\begin{aligned} &min \sqrt{(x_i - x_g)^2 + (y_i - y_g)^2 + (z_i - z_g)^2 + ...} \\ &\overline{\quad ... + (\alpha_i - \alpha_g)^2 + (\beta_i - \beta_g)^2 + (\gamma_i - \gamma_g)^2} \\ &\text{s.t.} \quad q_{min} < q_i < q_{max} \end{aligned} \tag{22}$$

To solve this minimization problem we use global and local algorithms from the open source NL-OPT library [16]. Although this solution of the inverse kinematics transformation has the advantage of including joint constraints, it still poses multiple issues.
The first issue is the computation effort needed by the solvers which represents a serious problem in real-time applications like ours. For this reason we have to run the optimization in a seperate thread. The other issue concerns the way global minimization techniques work. Because a global minimum search has to have a certain stochastic-evolutionary component to be able to escape local minima, these algorithms could deliver widely different joint configurations when the end cartesian goal is minimally shifted. This could lead to problems while trying to do online re-adjusting of the hitting point estimation by using newly acquired vision information. The new estimation of the hitting posture is accompanied with updates to the joint trajectory that is being executed. A large deviation of the end joint configuration would lead to a completely new joint trajectory with very little time to execute

which leads to high motor torques. The solve to this problem we suggest using the first inverse kinematic solution to initialize new instances of the optimization while re-adjusting. This allows the algorithm to converge faster and deliver joint configurations that are in the same region of the first solution.

6.4 Joint Trajectory Generation

After translating the hitting position into joint angles and velocities with the inverse kinematics, we have to plan a joint trajectory that provides the transition from the current position to hitting position and meets the time and speed constraints. The joint trajectory should also guarantee a smooth change in acceleration (related directly to motor torques). This would prevent jerky motor movements and improve the performance of the PD-Controller while following the trajectory. These conditions could be met with a fifth order polynomial for every joint [9]:

$$q_i(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + c_5 t^5 \tag{23}$$

The fifth order polynomial enables us to set boundary conditions on the velocities and acceleration for the start $t = 0$, and endpoint $t = T$. To get the parameters c_i we use the start and end conditions to solve the following system of linear equations:

$$\begin{aligned}
&q_i(0) = c_0 \qquad \dot{q}_i(0) = c_1 \qquad \ddot{q}_i(0) = 2c_2\\
&q_i(T) = c_0 + c_1 T + c_2 T^2 + c_3 T^3 + c_4 T^4 + c_5 T^5\\
&\dot{q}_i(T) = c_1 + 2c_2 T + 3c_3 T^2 + 4c_4 T^3 + 5c_5 T^4\\
&\ddot{q}_i(T) = 2c_2 + 6c_3 T + 12c_4 T^2 + 20c_5 T^3
\end{aligned} \tag{24}$$

6.5 Task Implementation

The pendulum SL task consists of multiple sections that are repeated sequentially every time a robot hits the ball. As a first step, we determine the ball's velocity components in spherical coordinates. As mentioned before we collect 5 samples and differentiate manually to estimate the speed the ball is moving with. This gives us all the necessary initial conditions to simulate the future trajectory that the ball will follow while undisturbed. We also use the samples to determine the direction of the pendulum's rotation around the Z-axis. Depending on that, one of the two robots is then activated and should hit the ball next.
Activation of the robot includes identifying a point on the trajectory where the paddle is supposed to intersect the ball's movement and calculating its inverse kinematic transformation. We choose to hit the ball on the point where it crosses the X-axis ($y = 0$). Now that we have the joint angles of the hitting posture, we initialize the minimal jerk joint trajectory with the current and goal joint information as well as the remaining time until intersection. The joint movement is then executed iteratively until we hit (or miss) the ball then the joint trajectory is reinitialized in order to go back to the default joint posture that we have started from. For this task we implement a simple hitting movement by setting an end speed for the (first) shoulder joint at the end of the joint trajectory. Other joint angles have zero velocity when the ball is intercepted. After finishing the joint movement the task starts collecting new vision samples to re-estimate the trajectory and activate the next robot. In Fig. 22 we illustrate the phases of intercepting the ball in simulation.

7 Tetherball Task

In this task we seek to implement the full tetherball setup in order for the robots to play autonomously. Because the tetherball model is fairly complex, but also similar that of the pendulum, we first tried to approximate the trajectory of the ball by fitting sine functions and ellipsoids. This lead to unreliable results considering the poor quality of the vision data.
Another approach was to regard the tetherball system as a pendulum with changing length and to fit the measured data into a pendulum model whose length is to be inferred. This has also lead to poor results because the models could be inferred only in discrete intervals and the extrapolation of those models did not give a good approximation.
Finally we derived the equations of motion analytically and we were able to acquire a minimally simplified model that delivered a good approximation of the ball's movement.

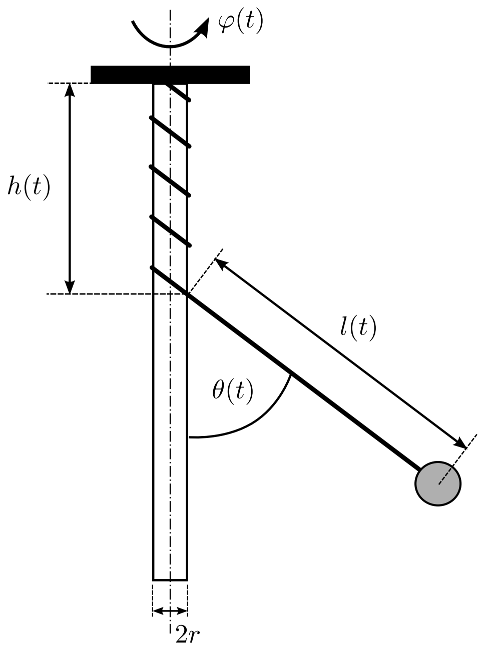


Figure 10: Tetherball Model

7.1 Tetherball Mechanical Model

In Fig. 10 we depict the tetherball schematic in spherical coordinates. The model has four states $\{\theta, \varphi, l, h\}$. The extra state h represents the vertical distance to the last contact point of the string with the pole. These four states are the generalized coordinates we use to arrive at the equations of motion.

A notable property of the system is that in the ball-pole fixed subsystem, the law of conservation of angular momentum does not apply. However the angular momentum is conserved in the the whole ball-pole-earth system. This is easily understood when considering that the tension force in the string exerts a torque around the center of the pole with a lever arm equal to the radius of the pole r. This torque - change in angular momentum - is lost to earth. The tetherball system is also a hybrid system meaning that it switches between two sets of differential equations. The transition happens when the absolute azimuth angle φ changes sign after winding and unwinding the string once. Another issue is that out of the four states only φ is measurable with our vision sensor. The polar angle θ is directly connected to l and h which can not to be determined easily.

The derivation of the equations of motion starts by writing down the mapping of cartesian coordinates to our generalized coordinates. For that we extend the transformation from the pendulum model to describe the string winding around the pole.

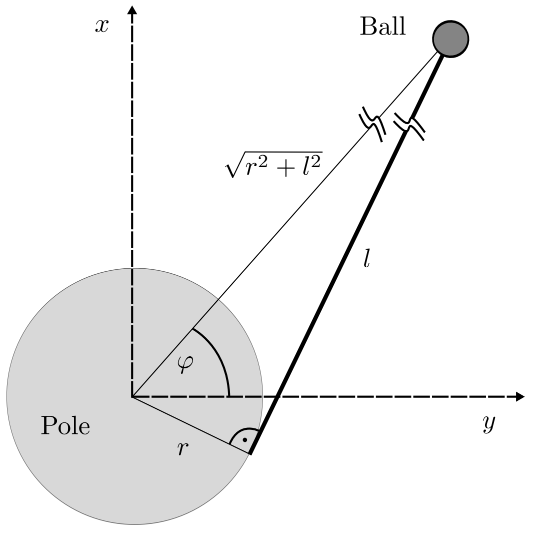


Figure 11: Planar Tetherball

$$\begin{aligned} x &\simeq l(t)\sin(\theta)\cos(\varphi) \\ y &\simeq l(t)\sin(\theta)\sin(\varphi) \\ z &= l(t)\cos(\theta) + h(t) \end{aligned} \tag{25}$$

Eq. 25 implements a simplified transformation as shown in Fig. 11. The simplification of the $\{x, y\}$ coordinates is based on ignoring the radius of the pole r when put in relation to the length of the string l. In addition to geometric equations, the model is constrained by two nonholonomic - not integrable - velocity conditions to $\{l, h\}$. These constraints are related to the nature of the model and the way the rope (un-)winds around the pole. We used Fig. 12 to formulate the constraints.

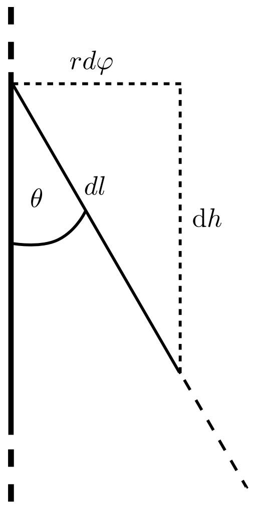


Figure 12: Differential Tetherball Model View

The resulting constraint equations are given by:

$$\begin{aligned} \dot{l} &= -\frac{r\dot{\varphi}}{\sin\theta} \\ \dot{h} &= \frac{r\dot{\varphi}}{\tan\theta} \end{aligned} \tag{26}$$

Because these constraints are not integrable, the Lagrangian Formalism we used for the pendulum model does not apply. We have to use the Lagrange-d'Alembert Principle as stated in [10]. For that we transform the constraints into vector form:

$$\sum_{k=1}^{n} a_k^j(q^i)\dot{q}^k = \mathbf{a}^T\dot{\mathbf{q}} = 0$$

$$\mathbf{a}^T\dot{\mathbf{q}} = \begin{bmatrix} 0 & r[\frac{1}{\tan\theta} - \frac{1}{\sin\theta}] & 1 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{\varphi} \\ \dot{l} \\ \dot{h} \end{bmatrix} \tag{27}$$

The Lagrange function consisting of the kinetic and potential energy is:

$$\begin{aligned} L &= T - U \\ T &= \frac{1}{2}m(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) \\ U &= -mgz \end{aligned} \tag{28}$$

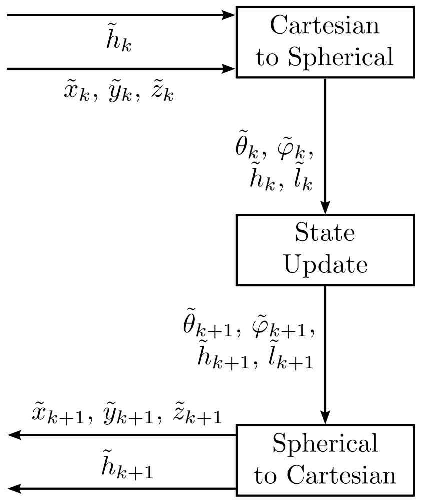
Now we can write out the Lagrange-d'Alembert equations:

$$\begin{aligned} \frac{d}{dt}\frac{\partial L}{\partial \dot{q}^i} - \frac{\partial L}{\partial q^i} &= \sum_{j=1}^{m} \lambda_j a_i^j \\ \frac{d}{dt}(\frac{\partial L}{\partial \dot{\theta}}) - \frac{\partial L}{\partial \theta} &= 0 \\ \frac{d}{dt}(\frac{\partial L}{\partial \dot{\varphi}}) - \frac{\partial L}{\partial \varphi} &= \lambda_1 \frac{r}{\sin\theta} - \lambda_2 \frac{-r}{\tan\theta} \\ \frac{d}{dt}(\frac{\partial L}{\partial \dot{l}}) - \frac{\partial L}{\partial l} &= \lambda_1 \\ \frac{d}{dt}(\frac{\partial L}{\partial \dot{h}}) - \frac{\partial L}{\partial h} &= \lambda_2 \end{aligned} \tag{29}$$

We notice that the d'Alembert Principle does not apply the constraints directly to the Lagrangian but to its displacement. To acquire the state space representation, we solve for $\{\ddot{\theta}, \ddot{\varphi}\}$. The resulting system of equations is too complex to be listed here. For that we provide a Wolfram Mathematica script that delivers the result of the calculation. It is also important to note that the resulting equations resemble only one part of the model namely the differential equations for a positive absolute φ. The second set of equations can be derived by editing the constraints to reflect a winding string with a negative absolute azimuth angle.

7.2 Vision and State Estimation

With the new equations of motion we face the issue of non-measurability of the spherical states. These states are needed for estimating the trajectory by integrating the initial conditions in a similar fashion to the pendulum task. Therefore we implement several state estimators like Particle Filter, Extended Kalman and Unscented Kalman in order to approximate the translation of the pivot along the pole h which is key to calculating the values of the other states.
Because we have 3 unmeasurable states out of 4, we cannot use the spherical state vector as it is, so we choose to use the cartesian state vector $\{x, y, z, h\}$ which contains more measurable information. The disadvantage of the cartesian representation is that it overcomplicates the equations of motion. We solve this problem by transforming the cartesian state vector to the spherical space then updating the system by using the spherical equations of motion and finally return to the cartesian space to compare with the measured cartesian vector, Fig. 13.



$\tilde{h}_k$
$\tilde{x}_k, \tilde{y}_k, \tilde{z}_k$
Cartesian to Spherical
$\tilde{\theta}_k, \tilde{\varphi}_k, \tilde{h}_k, \tilde{l}_k$
State Update
$\tilde{\theta}_{k+1}, \tilde{\varphi}_{k+1}, \tilde{h}_{k+1}, \tilde{l}_{k+1}$
$\tilde{x}_{k+1}, \tilde{y}_{k+1}, \tilde{z}_{k+1}$
$\tilde{h}_{k+1}$
Spherical to Cartesian

Figure 13: State Update Function

7.2.1 Particle Filter

We follow the general approach for the Particle Filter as described in [11]. Its structure, which we will describe in detail in the following section, is illustrated in Fig. 14.

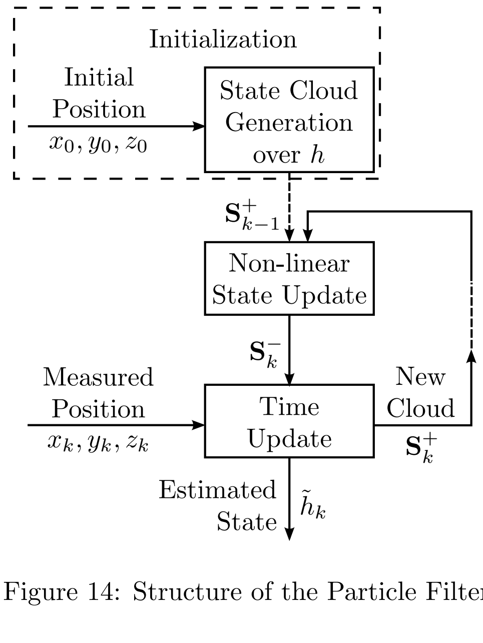


Figure 14: Structure of the Particle Filter

To *initialize* of the Particle Filter we take a cloud of points $\mathbf{S}_{k-1}^{+}$ ($N = 1000$) with the same cartesian states $\{x_0, y_0, z_0\}$[¶] and a uniform distribution over the state

[¶]Keep in mind that the state vector also contains the derivative of every generalized coordinate. We do not write out the derivatives for the sake of a compact representation

h:

$$\begin{aligned}\mathbf{x}_{0,i}^{+} &= \{x_{0,i}, y_{0,i}, z_{0,i}, h_{0,i}\} \\ &= \{x_0, y_0, z_0, h_i\} \\ \text{with} \quad & i = 1, ..., N.\end{aligned} \tag{30}$$

During the *state update* step we propagate $\mathbf{S}_{k-1}^{+}$ (composed of $\mathbf{x}_{k-1,i}^{+}$) through the nonlinear state update function to get the predicted state cloud $\mathbf{S}_{k}^{-}$ (composed of $\mathbf{x}_{k,i}^{-}$) as well as the predicted output vector $\hat{\mathbf{y}}_{k,i}$:

$$\begin{aligned}\mathbf{x}_{k,i}^{-} &= f(\mathbf{x}_{k-1,i}^{+}, w_{k-1,i}) \\ \hat{\mathbf{y}}_{k,i} &= h(\mathbf{x}_{k,i}^{-}, v_{k,i}),\end{aligned} \tag{31}$$

where $w_{k,i}$ and $v_{k,i}$ are randomly generated according to the known pdf of the process noise and the measurement noise.
Now for the *time update* we compare the predicted output $\hat{\mathbf{y}}_{k,i}$ to the actual measurements $\mathbf{y}_k = \{x_k, y_k, z_k\}$ from the Kinect sensor. This comparison is done by calculating the probability density function of predicted output in a gaussian distribution that has the measured state as its mean and measurement noise as variance. This gives us the conditional likelihood of each point in the cloud in relation to the current measurement:

$$\tilde{q}_i \sim \exp\left(-\frac{(\mathbf{y}_k - \hat{\mathbf{y}}_{k,i})^T \mathbf{R}^{-1} (\mathbf{y}_k - \hat{\mathbf{y}}_{k,i})}{2}\right) \tag{32}$$

After normalizing the likelihoods

$$q_i = \frac{\tilde{q}_i}{\sum_{j=1}^{N} \tilde{q}_j}, \tag{33}$$

we resample the updated state cloud for the best fitting points and get the new cloud $\mathbf{S}_k^{+}$. The sampling is done by first taking a uniformly distributed number α between 0 and 1, then calculating the cumulative sum of the normalized likelihoods sample by sample until it becomes greater than α and take the cloud point corresponding to the index we stopped at. This is repeated for N points. Thus we get a new cloud in which points with a higher likelihood q are more likely to be represented. However a too dominant point with very high likelihood could cause the filter to stagnate because it would be over represented in the newly sampled cloud. Now to obtain the estimated state vector (and hence also $\tilde{h}_k$) we average over all points of the new cloud $\mathbf{S}_k^{+}$.

In Fig. 18 we show the performance of the Particle Filter for both correct and incorrect initial velocities. The incorrect velocities were estimated from several sequential measurements, whereas the correct velocities were fitted offline. The result shows the filter's degree of sesnsitivity concerning the choice of the initial state cloud. A way to avoid the bad influence of the imprecise velocities would be to allow a distribution over $\dot{x}$, $\dot{y}$ and $\dot{z}$ in addition to that over h. However this would mean that we need a much larger cloud of points to cover three extra dimensions. This would lead to very unpractical computing times that do not suit the nature of our task.

7.2.2 Kalman Filter

Another way to estimate the state h is by implementing a Kalman Filter for the model of the tetherball. Because the system at hand is nonlinear we start by implementing the Extended Kalman Filter variation. In the following sections the state vector is of the form $\mathbf{x} = \{x, y, z, h\}$ with a starting guess for h_0. The measured state vector is $\mathbf{z} = \{x, y, z\}$. The following implementations of the Kalman Filters are based on [11].

Extended Kalman Filter: Like the Particle Filter the Extended Kalman Filter consists of the two steps *state prediction* and *state update*. In *state prediction* the filter predicts the future state vector $\widetilde{\mathbf{x}}_k$ based on the current states $\hat{\mathbf{x}}_{k-1}$ by integrating Eq. 18 at a rate of 30Hz. Belonging to the current state vector is the state covariance matrix $\widetilde{\mathbf{P}}_k$ that measures the accuracy of the state prediction. In order to predict the evolution of this matrix the system state update matrix $\mathbf{A}_{k-1}$ is needed and for that the state function f must be linearized at the working point. This is similar to evaluating the Jacobian at the current state. But because the Jacobian in this case is hard to determine analytically, we are forced to use fairly complicated and costly numerical methods like the Complex Step Jacobian or the Romberg Extrapolation Method. We set the process noise covariance matrix $\mathbf{Q}$ to zero assuming that our theoretical pendulum state update model has absolute accuracy, which is fairly plausible.

$$\begin{aligned}\widetilde{\mathbf{x}}_k &= f(\hat{\mathbf{x}}_{k-1}) \\ \mathbf{A}_{k-1} &= \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{k-1}} \\ \widetilde{\mathbf{P}}_k &= \mathbf{A}_{k-1}\mathbf{P}_{k-1}\mathbf{A}_{k-1}^T + \mathbf{Q}\end{aligned} \tag{34}$$

During *state update* the output vector of the former predictions $\widetilde{\mathbf{z}}_k$ is corrected based on the feedback comparison to current measurements $\mathbf{z}_k$. The vector $\widetilde{\mathbf{z}}_k$ is calculated by applying the output function h on the predicted state vector $\widetilde{\mathbf{x}}_k$. The correction term is proportional to the state prediction error. The correction intensity is defined by the Kalman gain $\mathbf{K}_k$ for which the Jacobian $\mathbf{H}_k$ of the output function and the measurement noise covariance matrix $\mathbf{R}$ are needed:

$$\begin{aligned}
\widetilde{\mathbf{z}}_k &= h(\widetilde{\mathbf{x}}_k) \\
\mathbf{H}_k &= \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\mathbf{x}=\widetilde{\mathbf{x}}_k} \\
\mathbf{K}_k &= \widetilde{\mathbf{P}}_k \mathbf{H}_k^T (\mathbf{H}_k \widetilde{\mathbf{P}}_k \mathbf{H}_k^T + \mathbf{R})^{-1} \\
\hat{\mathbf{x}}_k &= \widetilde{\mathbf{x}}_k + \mathbf{K}_k(\mathbf{z}_k - \widetilde{\mathbf{z}}_k) \\
\mathbf{P}_k &= \widetilde{\mathbf{P}}_k - \mathbf{K}_k \mathbf{H}_k \widetilde{\mathbf{P}}_k
\end{aligned} \tag{35}$$

After testing the Extended Kalman Filter in MATLAB we were confronted with stability issues because of the high nonlinearity of the system. Attempts to implement the algorithm in C proved also to be demanding because of the difficulty in the numerical calculation of the Jacobian. For example the realization of the Complex Step Jacobian algorithm, which is fast and provides improved stability, requires the differential equations to be solved with complex entries. Considering that the GSL library lacks direct support on this matter, we would have had to split the differential equations to real and imaginary parts, which would have increased the number and the complexity of the equations. For these reasons we moved to implementing the Unscented Kalman Filter whose algorithm does not include linearizing the system at the current working state resulting in better stability and removing the numerical obstacles concerning the calculation of the Jacobian.

Unscented Kalman Filter: This implementation of the filter is based on the Unscented Transform where fixed points, referred to as *sigma points*, representing a probability distribution are propagated through a nonlinear function to calculate the transformed distribution. As with other Kalman Filter variations the algorithm starts with a *state prediction.* The first step is to create $2n+1$ sigma points around the last Kalman state vector $\hat{\mathbf{x}}_{k-1}$ by sampling an initial distribution:

$$\begin{aligned}
\hat{\mathbf{X}}_{k-1} &= [\hat{\mathbf{x}}_{k-1} \quad \hat{\mathbf{x}}_{k-1} \quad \hat{\mathbf{x}}_{k-1}] + ... \\
&... + \sqrt{c}[0 \quad \sqrt{\mathbf{P}_{k-1}} \quad -\sqrt{\mathbf{P}_{k-1}}]
\end{aligned} \tag{36}$$

The constant c is defined by the number n of states and two tuning parameters:

$$c = \alpha^2(n+\kappa) \tag{37}$$

These sigma points are then propagated through the nonlinear state update function f to result in a new distribution defined by the predicted mean $\widetilde{\mathbf{x}}_k$ and predicted state covariance matrix $\widetilde{\mathbf{P}}_k$:

$$\begin{aligned}
\mathbf{X}_k^- &= f(\hat{\mathbf{X}}_{k-1}) \\
\widetilde{\mathbf{x}}_k &= \mathbf{X}_k^- \mathbf{w}_m \\
\widetilde{\mathbf{P}}_k &= (\mathbf{X}_k^- \mathbf{M}) \mathbf{W}_c (\mathbf{X}_k^- \mathbf{M})^T + \mathbf{Q}
\end{aligned} \tag{38}$$

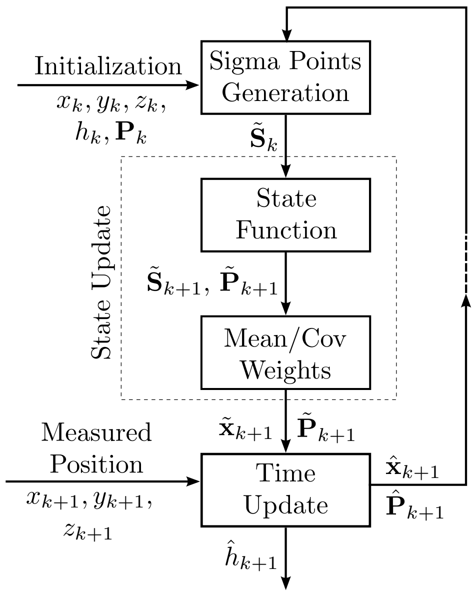


Figure 15: Structure of the Unscented Kalman Filter

$\mathbf{W}_c$ and $\mathbf{w}_m$ are the covariance weights matrix and the mean weights vector which reflect the importance of the original Kalman state compared to the sigma points that were created around it in Eq. 36:

$$\begin{aligned}
W_m^{(0)} &= (c-n)/c \\
W_m^{(i)} &= 1/2c \\
\mathbf{w}_m &= [W_m^{(0)} \quad ... \quad W_m^{(2n)}]^T \\
\mathbf{M} &= \mathbf{I} - [\mathbf{w}_m \quad ... \quad \mathbf{w}_m]
\end{aligned} \tag{39}$$

$$\begin{aligned}
W_c^{(0)} &= (c-n)/c + (1-\alpha^2+\beta) \\
W_c^{(i)} &= 1/2c \\
\mathbf{W}_c &= diag([W_c^{(0)} \quad ... \quad W_c^{(2n)}])
\end{aligned} \tag{40}$$

where β is the third Kalman tuning parameter that determines the distribution of the sigma points. For a Gaussian distribution β is optimally chosen to be 2. The *state update* comes to correct the first prediction by comparing the predicted mean output $\widetilde{\mathbf{z}}_k$ to the measured output $\mathbf{z}_k$. $\widetilde{\mathbf{z}}_k$ is calculated by propagating newly generated sigma points from the predicted distribution through the nonlinear output function h:

$$\begin{aligned}
\widetilde{\mathbf{X}}_k &= [\widetilde{\mathbf{x}}_k \quad \widetilde{\mathbf{x}}_k \quad \widetilde{\mathbf{x}}_k] + ... \\
&... + \sqrt{c}[0 \quad \sqrt{\widetilde{\mathbf{P}}_k} \quad -\sqrt{\widetilde{\mathbf{P}}_k}] \\
\widetilde{\mathbf{Z}}_k &= h(\widetilde{\mathbf{X}}_k) \\
\widetilde{\mathbf{z}}_k &= \widetilde{\mathbf{Z}}_k \mathbf{w}_m
\end{aligned} \tag{41}$$

The correction term and the Kalman gain are calculated similar to the Extended Kalman Filter and the predicted mean and covariance matrix are updated:

$$\begin{aligned}\mathbf{S}_k &= (\widetilde{\mathbf{Z}}_k\mathbf{M})\mathbf{W}_c(\widetilde{\mathbf{Z}}_k\mathbf{M})^T + \mathbf{R}\\ \mathbf{C}_k &= (\widetilde{\mathbf{X}}_k\mathbf{M})\mathbf{W}_c(\widetilde{\mathbf{Z}}_k\mathbf{M})^T\\ \mathbf{K}_k &= \mathbf{C}_k\mathbf{S}_k^{-1}\\ \hat{\mathbf{x}}_k &= \widetilde{\mathbf{x}}_k + \mathbf{K}_k(\mathbf{z}_k - \widetilde{\mathbf{z}}_k)\\ \mathbf{P}_k &= \widetilde{\mathbf{P}}_k + \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T\end{aligned} \tag{42}$$

In Fig. 19 we show the performance of the Unscented Kalman Filter. It is clear that this filter performs better than the Particle Filter and the estimation error for the state h in this sample goes down to approximately 5cm. Another comparison of the two filters showing the evolution of state error over time is to be found in Fig. 20.

7.3 Task Implementation

The implementation of the tetherball task with the Unscented Kalman filter is in its structure more complicated than the pendulum task. The Unscented Kalman filtering and the trajectory estimation run in a separate thread because they perform heavy numerical calculations that would slow the main thread down. The Unscented Kalman is running constantly and is only reset when the ball changes the direction of rotation or when the filter becomes unstable.
We estimate the trajectory about 3 times a second by integrating the current Kalman states. After each trajectory estimation we activate one robot to hit the ball and keep updating the trajectory until shortly before the intersection point. Meanwhile in the main SL thread we choose a hitting point ($y = 0$) on the trajectory, calculate the joint configuration using the inverse kinematics and initialize the minimal jerk joint trajectory. When the secondary thread updates the trajectory of the ball the previous sequence is run again and the joint trajectory is updated until we hit the ball. The continuous correction of the interception point, which is shown in Fig. 21, improves the accuracy dramatically and allows for quick reactions in case the robot misses the ball on its first try. During the task we use the absolute azimuth angle φ to detect the switch modes in the mechanical equations of motion. This is crucial in order for the Kalman filter and trajectory estimation to work.

8 Conclusion

In this paper we presented a new platform for playing tetherball between two BioRob robotic arms. We first described the physical setup and the communication structure between all setup components. We have also successfully implemented a new algorithm for detecting the moving ball despite the shortcomings of the Kinect. In addition we created a vision calibration task in order to infer the spatial transformations between the robots, the Kinect and the ball. We then went on to implement a simplified pendulum task that helped us in testing our solutions to problems like inverse kinematic and minimum jerk trajectories. For the tetherball task we derived the equations of the ball's movement by using the Lagrange-d'Alembert Principle. However the resulting the state vector was mostly unmeasurable so we tested multiple state estimators and achieved the best result by using the Unscented Kalman Filter. Finally we assembled all the subtasks into one algorithm and were able to reach our goal of playing tetherball with compliant robots.

Acknowledgment

We would like to thank our advisors Prof. Jan Peters and Alexandros Paraschos of the Institute for Intelligent Autonomous Systems at the Technical University of Darmstadt for their valuable input and support during the project.

References

[1] K. Muelling, J. Peters, *A computational Model of Human Table Tennis for Robot Application.* Proceedings of Autonome Mobile Systeme, 2009.

[2] K. Muelling, J. Kober, J. Peters, *Learning Table Tennis with a Mixture of Motor Primitives.* Proceedings of the International Conference on Humanoid Robots, 2010.

[3] Z. Wang, C. Lampert, K. Muelling, B. Schoelkopf, J. Peters, *Learning to Select and Generalize Striking Movements in Robot Table Tennis.* Proceedings of the International Conference on Intelligent Robot Systems, 2011.

[4] J. Peters, J. Kober, K. Muelling, O. Kroemer, G. Neumann *Towards Robot Skill Learning: From Simple Skills to Table Tennis.* Proceedings of the European Conference on Machine Learning, 2013.

[5] C. Daniel, G. Neumann, J. Peters *Learning Concurrent Motor Skills in Versatile Solution Spaces.* Proceedings of the International Conference on Robot Systems, 2013.

[6] J. Kober, M. Glisson, M. Mistry, *Playing Catch and Juggling with a Humanoid Robot.* Proceedings of the International Conference on Humanoid Robots, 2012.

[7] O. von Stryk, *Foundations of Robotics.* TU-Darmstadt, 2014.

[8] O. von Stryk, *Optimization of Static and Dynamic Systems.* TU-Darmstadt, 2014.

[9] M. Spong, S. Hutchinson, M. Vidyasagar, *Robot Modeling and Control*. Wiley a. Sons, 2005.

[10] A.M. Bloch, *Nonholonomic Mechanics and Control*. Springer, 2003.

[11] D. Simon, *Optimal State Estimation*, Wiley a. Sons, 2006.

[12] S. Schaal, *The SL Simulation and Real-Time Control Software Package*. http://www-clmc.usc.edu/publications/S/schaal-TRSL.pdf, 2006.

[13] OpenCV Developers Team, *OpenCV Documentation*. http://docs.opencv.org/, 2014.

[14] M.I.A. Lourakis, *Levenberg-Marquardt Nonlinear Least Squares Algorithms in C/C++*. http://www.ics.forth.gr/lourakis/levmar/+, 2004.

[15] The GNU Operating System, *GNU Scientific Library*. http://www.gnu.org/software/gsl/, 2013.

[16] Steven G. Johnson, *The NLopt nonlinear-optimization package*. http://ab-initio.mit.edu/nlopt, 2014.

[17] Orocos Real-Time Toolkit, *RTT Online API Documentation*. http://www.orocos.org/stable/documentation/rtt/v2.6.x/api/html/index.html, 2014.

[18] Open Source Robotics Foundation, *Documentation - ROS Wiki*. http://wiki.ros.org/, 2013.

[19] The Internet Society, *An Introduction to the Stream Control Transmission Protocol (SCTP)*. http://tools.ietf.org/html/rfc3286, 2002.

Appendix

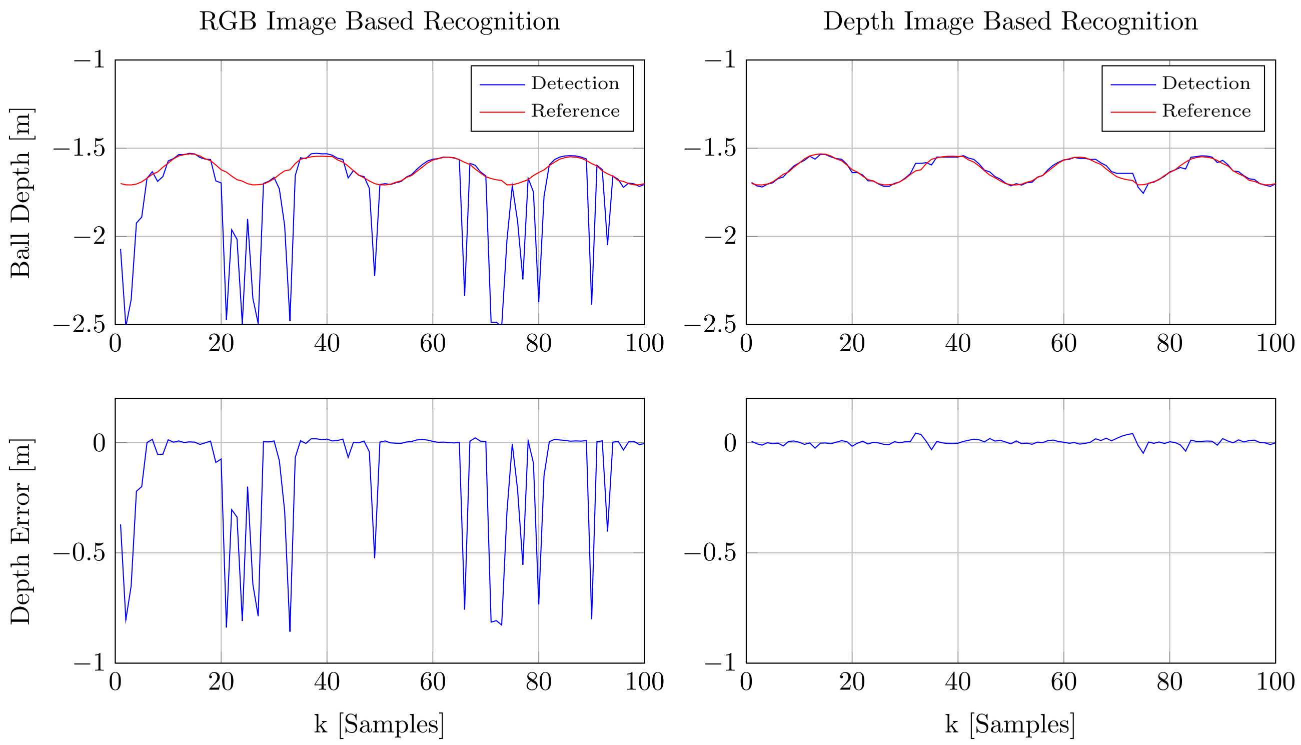


Figure 16: Comparison of RGB Based Detection and Depth Based Detection

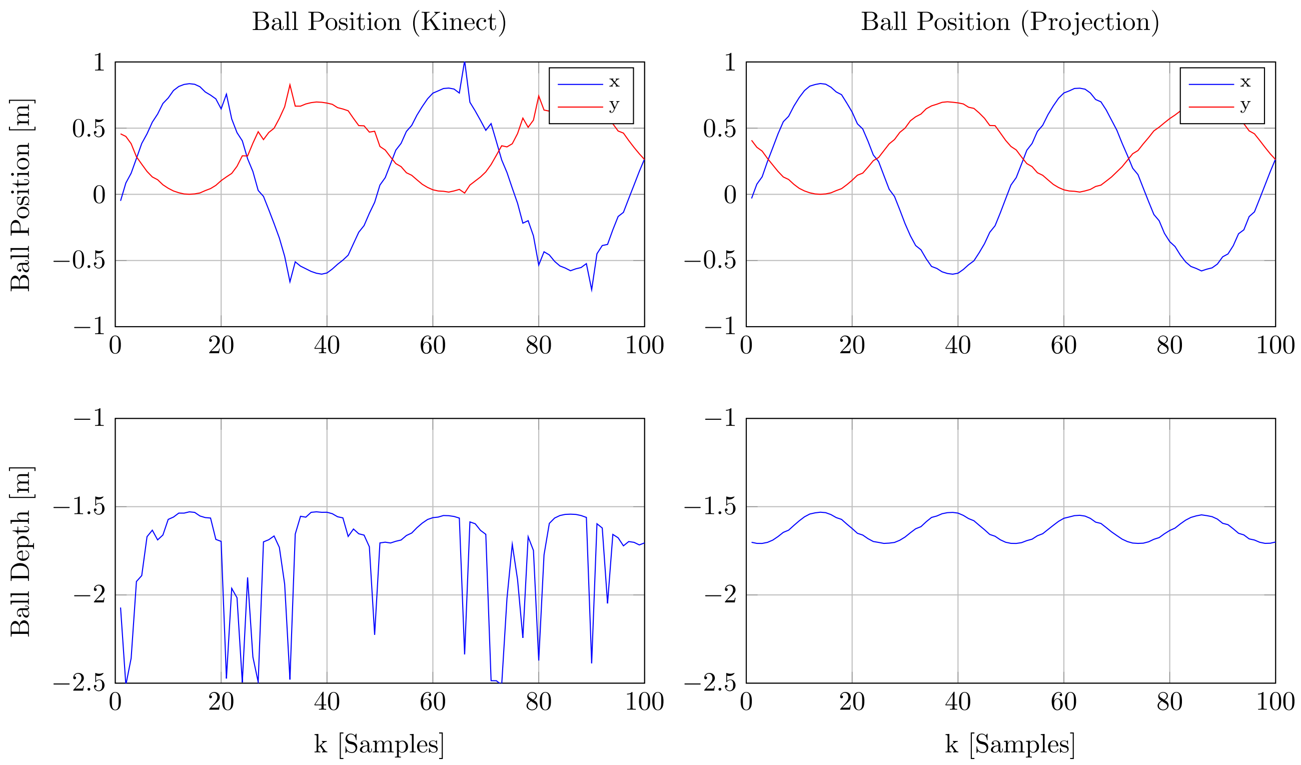


Figure 17: Comparison of Ball Position from Kinect and Ball Position Transformed on Sphere

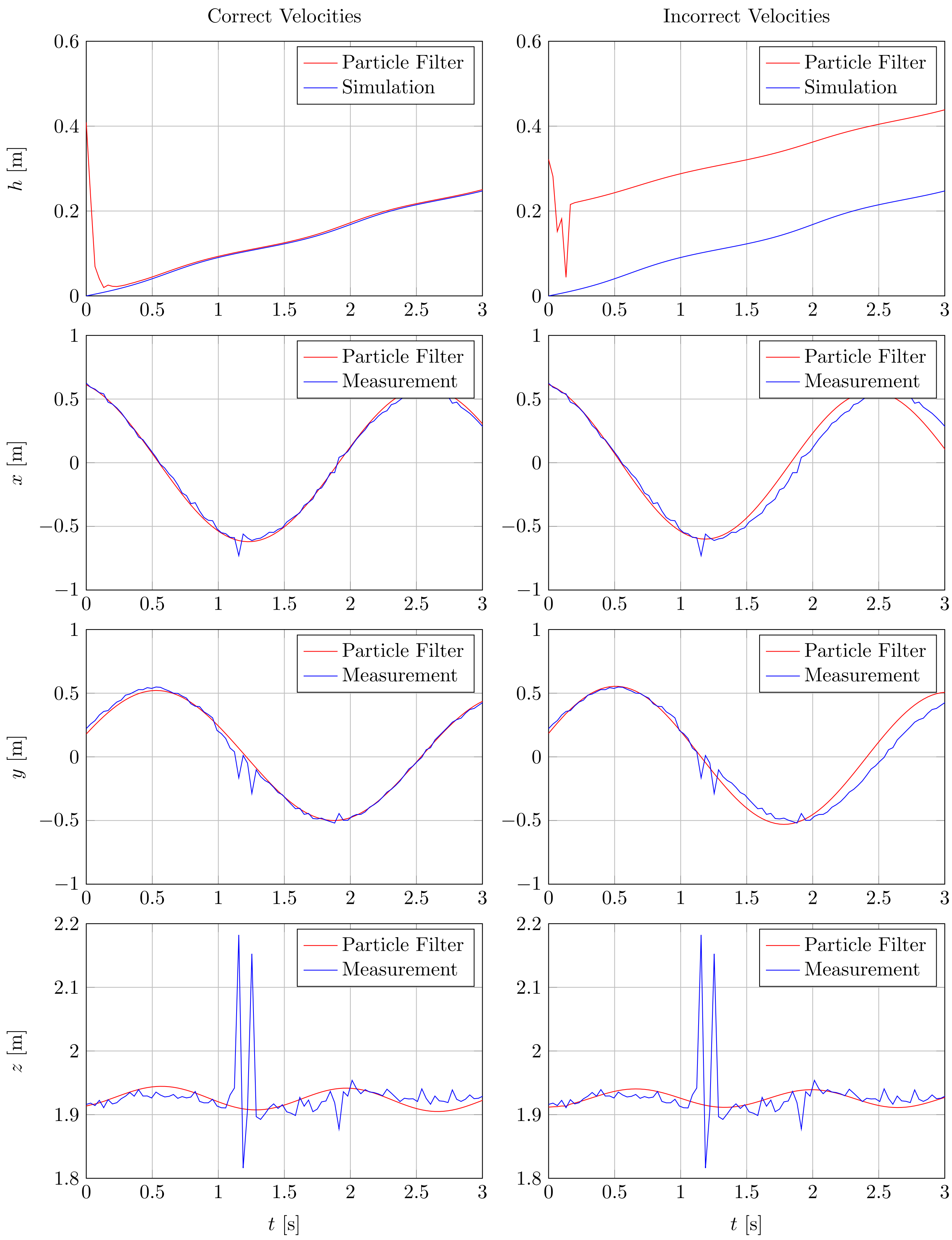


Figure 18: Performance of Particle Filter with Correct and Incorrect Velocities

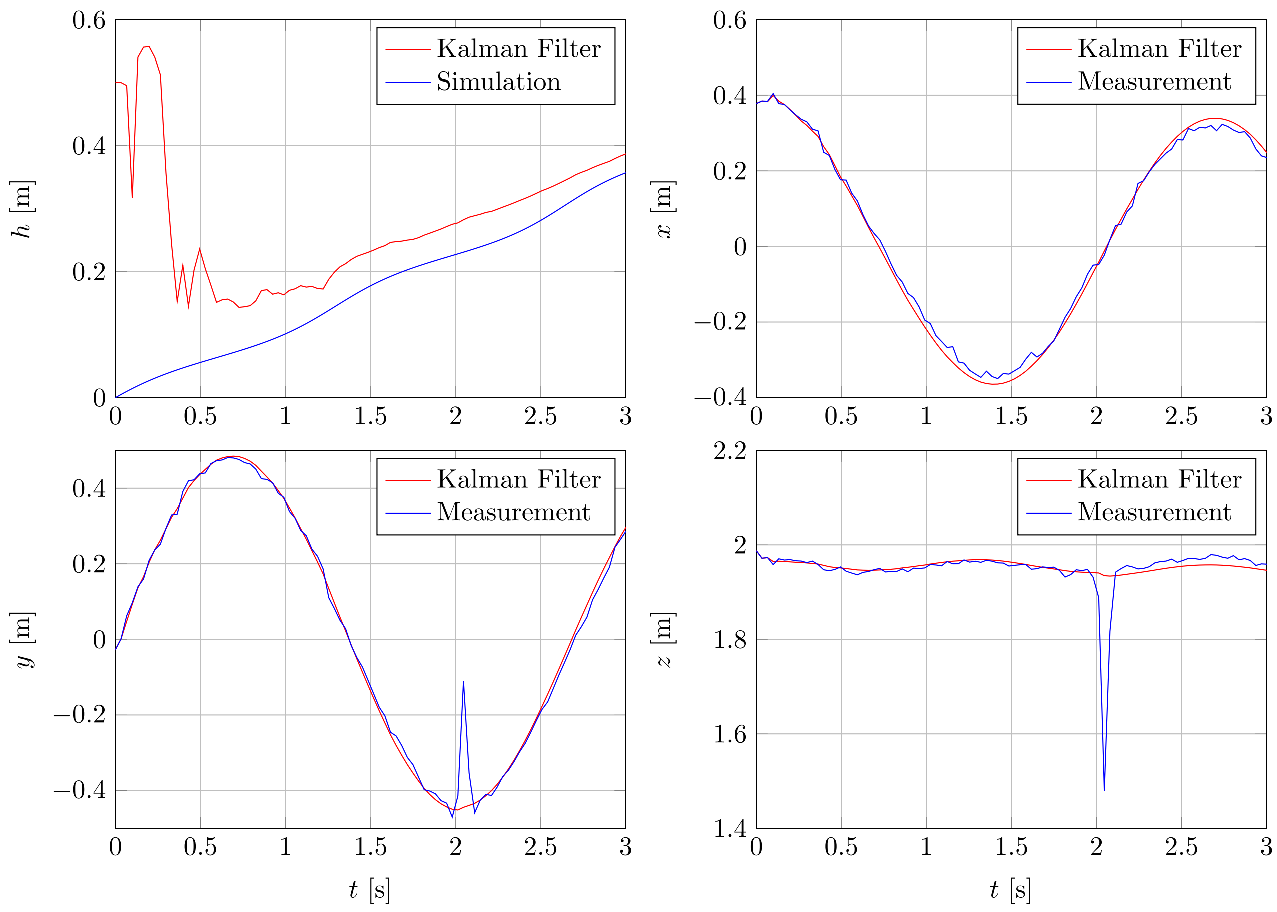


Figure 19: Performance of Unscented Kalman Filter

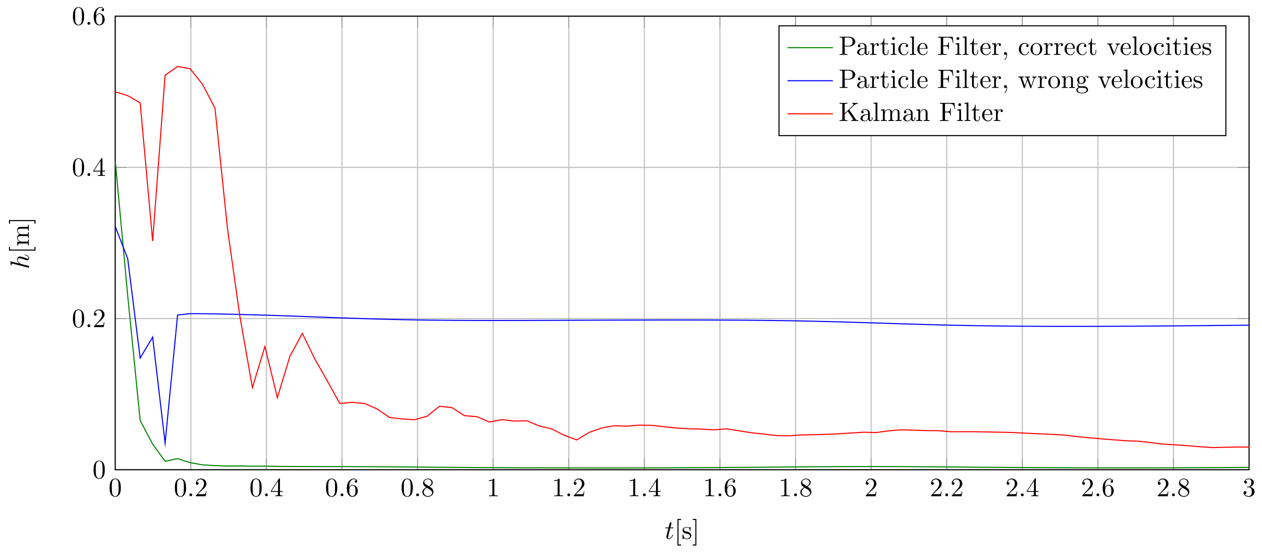


Figure 20: Comparison of Performance of Particle Filter and Unscented Kalman Filter

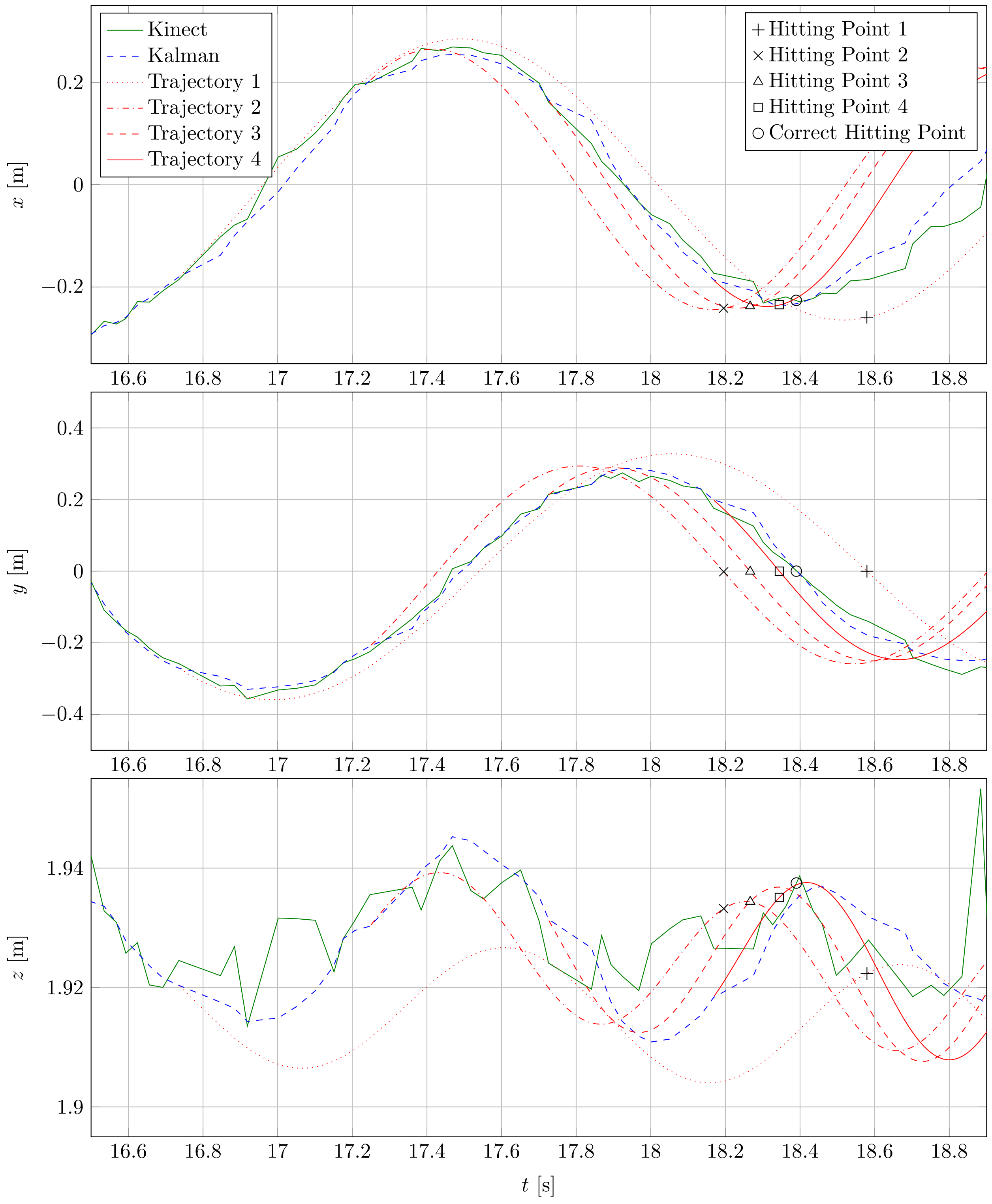
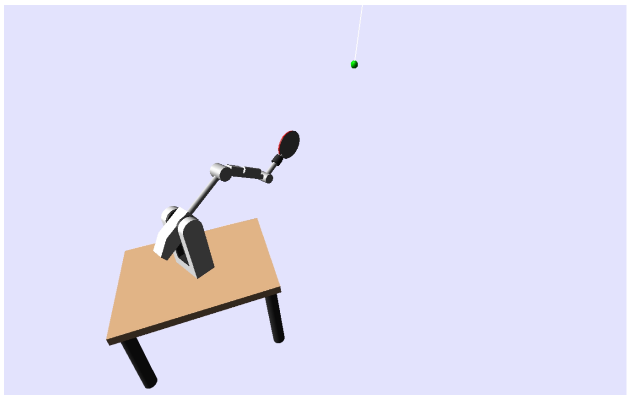
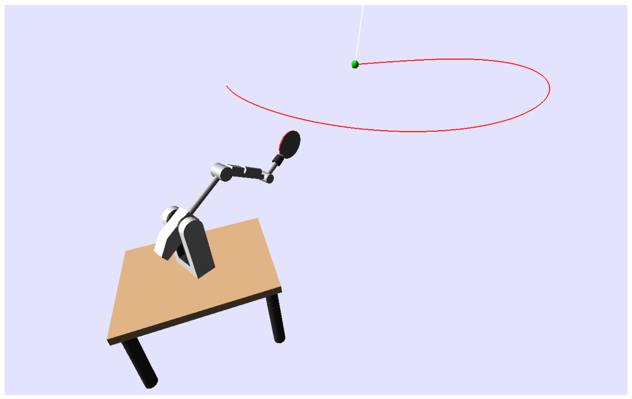
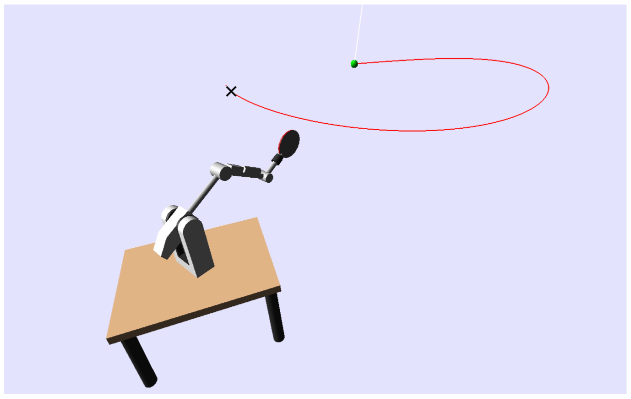
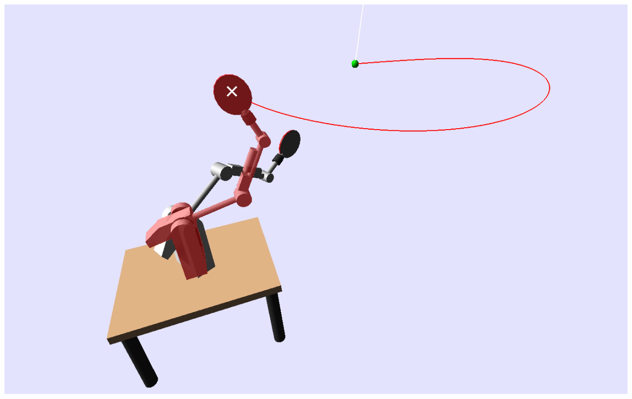


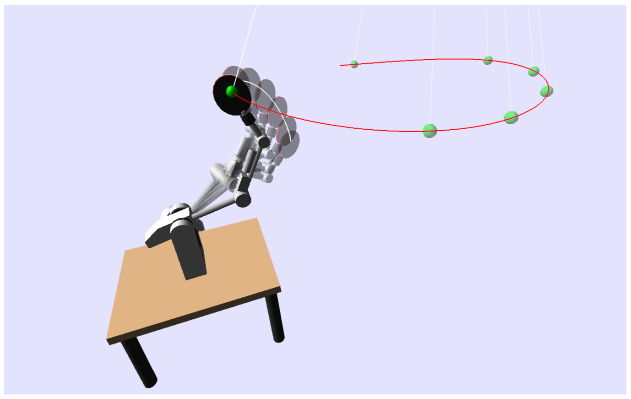
Figure 21: Multiple Correction of Trajectory Estimation and Resulting Hitting Points


(a) Measure Position and Estimate Velocities


(b) Integrate Initial State Using Model Equations


(c) Choose an Interception Point on the Trajectory ($y = 0$)


(d) Apply Inverse Kinematics


(e) Minimal Jerk Joint Trajectory

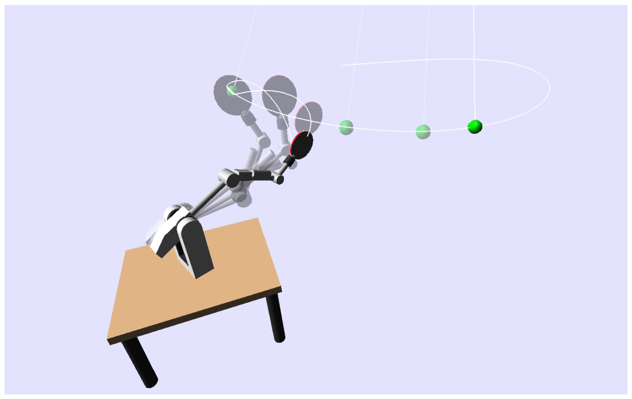

(f) Back to Default Posture

Figure 22: Implementation of the Pendulum Task