Lab 5

Closed Loop Control

Author:

Jonas Lussi, Daniel Steinmann, Alexandre Mesot, Michelle Mattille, Naveen Shamsudhin, and Prof. Bradley J. Nelson

Institute of Robotics and Intelligent Systems

Date: 2022

Version: 1.0

Summary: The purpose of this week's lab is to familiarize with inverse kinematics and camera calibration. You will complete the first necessary steps towards programming a fully functional Ball Balancing Robot.

5.1 Introduction

5.1.1 Background

In this lab you can apply your obtained knowledge to a ball balancing system. It consists of 3 servo motors (similar to the motors that you have seen in Lab 04), joints that connect the motors to a plate, as well as a camera for visual feedback. The general setup is depicted in 5.1 The logical flow for the control of the system can be seen in Fig. 5.3. To drive the plate that balances the ball, we use three servo motors that are connected to the plate via three separate two-link arms. This allows us to control the two rotational degrees of freedom of the plate, as well as its height. In this lab you will focus on understanding the kinematics and camera calibration.

Note: The design of this system was inspired by Johan Link's work: `https://github.com/JohanLink/Ball-Balancing-PID-System`

5.1.2 Feedback Control

The goal of this week's lab is get the system up and running so that we can implement the PID controller in the next lab. We will quickly go through the elements of a closed-loop control system to understand which preparations are necessary before starting to implement the control algorithm in C. In Figure 5.2 you can see the flowcharts of an open-loop and a closed-loop system. If you are not familiar with these two types of control systems, you can find many great examples here. The *output* of our system is the ball's position. We will use a camera and a calibration procedure to obtain the physical coordinates of the ball. The *inputs* are the tilt angles of the plate. The servo motors are driven by commands which are computed using the inverse kinematics algorithm we will derive in the prelab. To sum up, before we can start implementing the *controller*, we have to set up the plate and the camera in order to have reliable inputs and outputs.

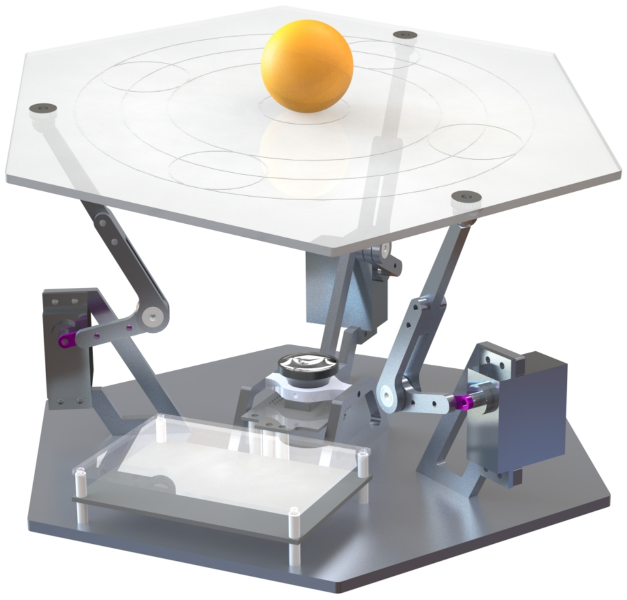


Figure 5.1: A rendered picture of the ball balancing system

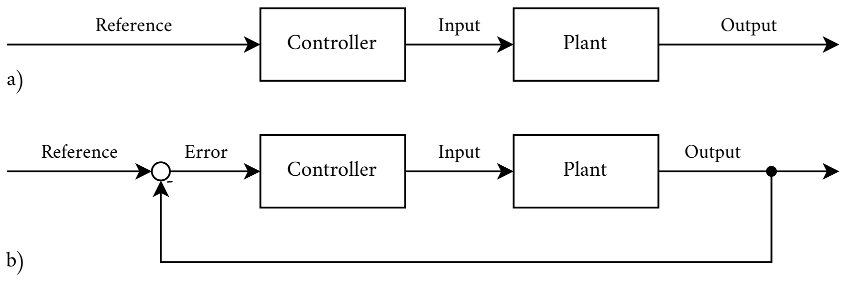


Figure 5.2: a) Open loop control. The system output is not measured. b) Closed loop control. The system output is measured and compared against the reference value. The controller can react to unexpected errors and correct the input to the system accordingly.

5.2 System Description

Figure 5.3 provides an overview over the system components, which are described in the following paragraphs.

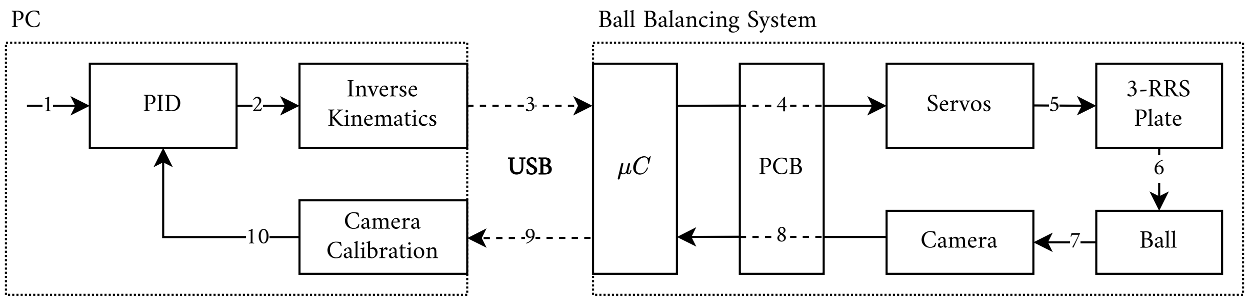


Figure 5.3: System components and signal connections. 1) Ball reference coordinates, 2) Plate angles, 3) Servo angles as string 4) Servo angles as PWM signal, 5) Mechanical connection between servo and legs, 6) Contact between ball and plate, 7) Camera observes ball and computes coordinates, 8) Pixel coordinates as SPI signal, 9) Pixel coordinates as string, 10) Calibrated coordinates.

5.2.1 Pixy2 Camera

The Pixy2 (Figure 5.4) is a camera with integrated object tracking that ships at about 80 \$. The on-board NXP LPC4330 processor applies an optical flow algorithm to the image stream provided by the Aptina MT9M114 image sensor. Up to 7 objects can be tracked simultaneously, and their coordinates are updated at a rate of 60 Hz. When connected to a PC via USB, the Pixy2 can be calibrated with the graphical interface PixyMon. Once the calibration is finished, the USB connection is removed, and the Pixy2 remains connected only to the μC via an SPI connector. The μC can then request the current coordinates without the need for additional image processing. More details can be found on the provided datasheet.

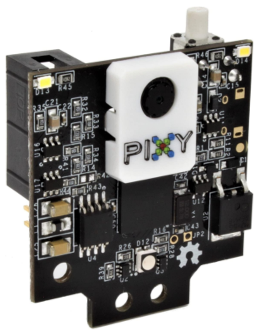


Figure 5.4: The Pixy2 camera by CharmedLabs. The button on the top right allows to train objects and adjust the white-balance without the USB connection. Feedback is provided with color and intensity of the LED, more information is provided in the documentation.

5.2.2 Servos

For the mechanical actuation, we use three RC-Servos of type KST DS725MG, see Figure 5.5. The servos feature a metal gear unit, can be operated at a supply voltage at up to 8.4 V and provide a stall torque of 18.0 kg/cm. At a price of ca. 50 \$, the servos belong to the high-performance class of RC-servos particularly due to their high speed of up to 0.42 s/rev. This high speed allows us to model the plate movements as instantaneous with a negligible delay, which would be more problematic when using slower motors. Of course, an unmodeled delay remains and has to be kept in mind. More details can be found in the provided datasheet.

Figure 5.5: RC-Servo KST DS725MG

5.2.3 PID

The PID controller compares the reference position provided by the user with the physical coordinates coming from the camera. In the next lab, you will learn how to design a PID controller, test it in Matlab and implement it in C.

5.2.4 Microcontroller (μC) and PCB

As for the previous labs, we use a Feather HUZZAH ESP32, which we call Featherboard. It is mounted on a custom-made printed circuit board (PCB) and is connected to the PC via USB.

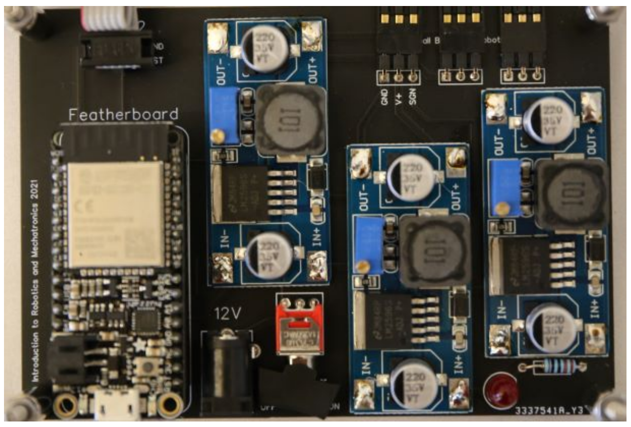


Figure 5.6: Photo of the PCB. The Featherboard is mounted on the left and can be replaced without the need for soldering.

5.2.5 3-RRS Plate

Parallel Mechanism

The servos are connected to three legs with two links each. The two links are connected with a knee-joint, and the second link connects to the plate with a magnetic spherical joint. This setup is a *parallel mechanism* of type 3-Revolute-Revolute-Spherical (3-RRS). It allows having an unobstructed view from the camera to the plate. Parallel mechanisms are generally harder to treat than serial mechanism, because the relation between the servo angles and the plate's position and orientation and the servo angles, called the *kinematics*, are generally more complicated. In subsection 5.2.7, we will take a closer look at the inverse kinematics problem.

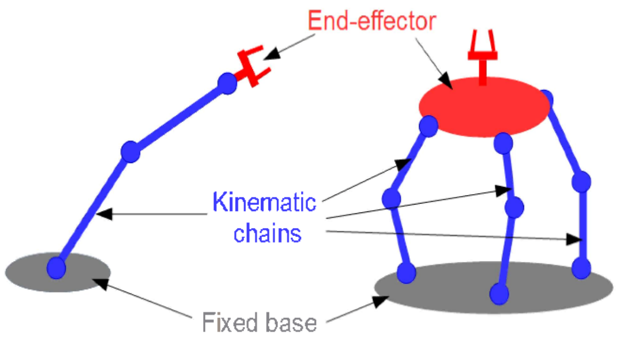


Figure 5.7: Serial (left) vs. parallel mechanisms (right). The end effector of a parallel mechanism is directly connected to its base by a number of separate and independet linkages working simultaneously.

Plate

The plate is made out of frosted acrylic. This material attenuates external lighting disturbances, while the shape of the ball is still visible to the camera. The engravings which serve as an external optical reference for the ball position are shown in Figure 5.8.

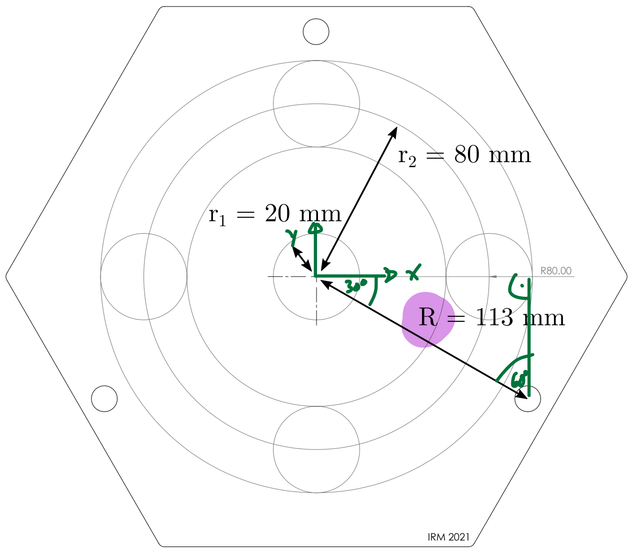


Figure 5.8: Top plate with engravings. Five circles with radius r_1 can be used to evaluate the position tracking accuracy of the controller. The large circle with r_2 serves as a reference for a circular trajectory. R describes the distance from the plate center to the joint centers.

5.2.6 Camera Calibration

In contrast to for example a touchscreen or touch panel, the coordinates provided by a camera need to be calibrated to accurately represent the ball's position in world frame. Calibration of the camera is an essential part in most robotic applications that include vision. Matlab's image processing toolbox contains a camera calibration application, which computes the intrinsic and extrinsic camera parameters based on a set of checkerboard images. For its application, no in-depth knowledge about the algorithm is required, which is why it is not included in the scope of this lab. However, once the parameters are known, you are expected to implement the transformation to the camera coordinates that are described below.

All coordinates are depicted in Figure 5.9. The coordinates $\vec{u}' = [u', v']$ represent the coordinates in pixel returned by the camera. First, the radial distortion has to be removed from the pixel coordinates. $\vec{u}'_c = [u'_c, v'_c]$ describes the radial distortion center (or "principal point"), k_1 and k_2 are the radial distortion parameters. The correction is applied to normalized coordinates $\tilde{\vec{u}}'$, which are obtained by subtracting the distortion center and dividing the pixel coordinates by the focal length in pixel.

Normalized:
$$\tilde{\vec{u}}' = \frac{\vec{u}' - \vec{u}'_c}{f} \tag{5.1}$$

Radial distortion depends on the distance to the distortion center, so we compute a normalized radius:

$$\tilde{r} = |\tilde{\vec{u}}'| = \sqrt{\tilde{u}'^2 + \tilde{v}'^2} \tag{5.2}$$

We obtain the undistorted normalized coordinates as follows:

$$\tilde{\vec{u}} = \tilde{\vec{u}}' \frac{1}{1 + k_1\tilde{r}^2 + k_2\tilde{r}^4} \tag{5.3}$$

With the correction now applied, we rescale to obtain pixel values again, undistorted this time.

$$\vec{u} = f\tilde{\vec{u}} \tag{5.4}$$

From the pin-hole model, we know how to relate the pixel coordinates $\vec{u}$ to coordinates in the camera frame $\vec{x}_k$. The plate height P_z is known from the inverse kinematics, so we can compute z as the sum

$$z = |\vec{r}_{WK,z}| + P_z + r_{PP}, \tag{5.5}$$

where r_{PP} is the radius of the ping-pong ball and $\vec{r}_{WK,z}$ is the offset between the camera and the origin plane. Now we can use z to compute the camera frame coordinates:

$$\vec{x}_k = \vec{u}\frac{z}{f} \tag{5.6}$$

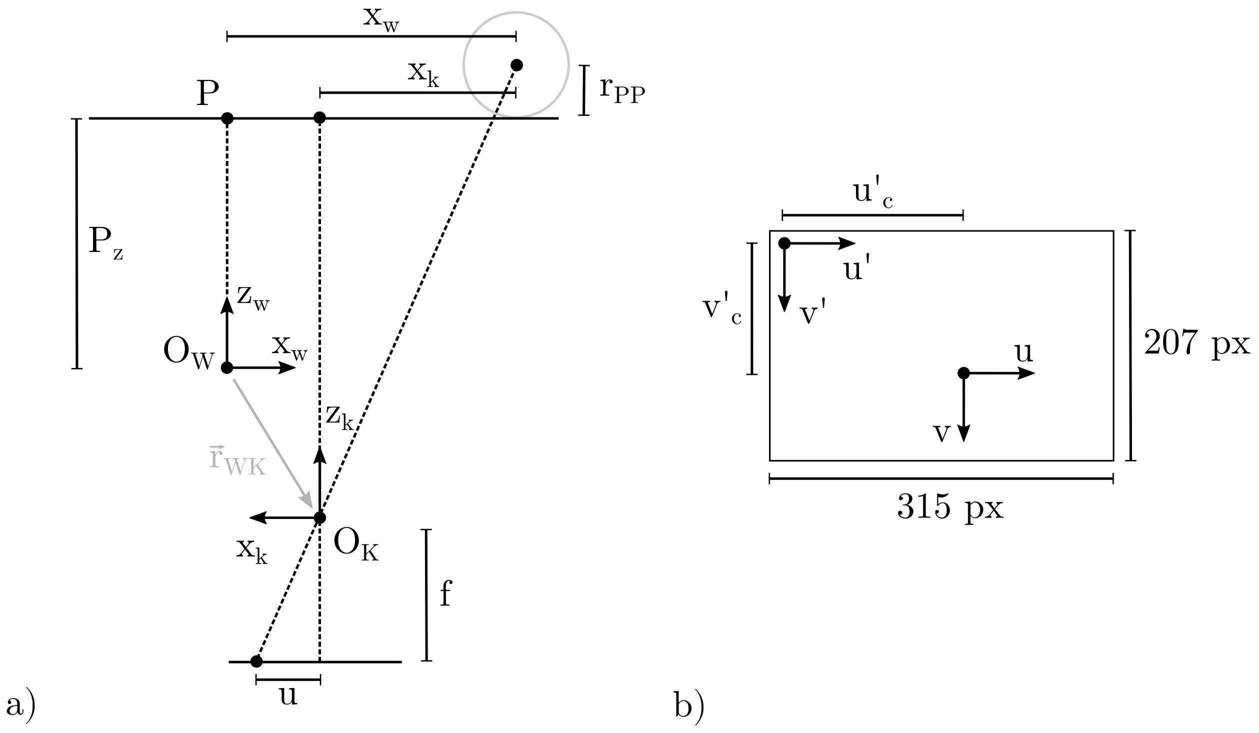


Figure 5.9: a) Camera calibration overview in 2D. The focal point O_K of the camera is the origin of the camera coordinate system (x_k, y_k, z_k). The focal length f is an intrinsic parameter, while vector $\vec{r}_{WK}$ represents the extrinsic translation between the camera frame O_K and the world frame O_W.The offset $r_{WK,z}$ accounts for the height difference between O_W-plane and the placement of the camera. The camera frame O_K is also rotated about the z-axis by $\psi_z = \pi$ w.r.t O_W. The pixel coordinate u relates to the physical x_k coordinate using a pin-hole projection model, and is translated to x_w using the extrinsic calibration parameters. This also applies to the y-coordinate. The z-coordinate is known using the inverse kinematics for P and $\vec{r}_{WK}$
b) The received pixel coordinates (u', v') originate on the top left. Using the distortion center (or "principal point") (u'_c, v'_c) and the distortion parameters k_1, k_2, radial distortion is removed from the pixel coordinates. Renormalizing by f yields the centered pixel coordinates (u, v).

At last, we need to rotate the camera frame to match the world frame orientation, and we need to add the camera frame offset:

$$\vec{x}_w = -\vec{x}_k + \vec{r}_{WK,xy} \tag{5.7}$$

The camera frame offset has to be found by experiment, since it depends on the mechanical setup and the alignment accuracy of the camera and plate. Keep in mind, we assume that the distortion center coincides with the center of the plate in the undistorted images after the calibration.

Note: When taking pictures with the Pixy2, the image size depends on the window size of PixyMon. Matlab then computes the calibration parameters using this window size, which has to be considered when implementing the calibration. We computed a constant scale factor λ between the image size and the actual coordinate range, which is premultiplied on every pixel value to match the scale of the calibration images.

5.2.7 Inverse Kinematics

The inverse kinematics are the set of equations which relate the desired tilt angles of the plates to the servo angles. As opposed to forward kinematics, which yield the plate orientation for a given set of servo angles, the inverse kinematics perform the opposite task: The PID will provide the necessary plate angles, which then have to be attained using the motors. For this, we first have to define a coordinate system and all the necessary angles, which are visualized in Figure 5.10. The coordinate system origin O is placed in the center of the plane which coincides with the servo joint axes, as shown in Figure 5.10. The x- and y axes are aligned with two of the base plate's axes of symmetry and the z-axis points upwards. To describe the top plate's orientation with respect to the base coordinate system, we use Euler angles: We assume that the yaw angle ψ_z is zero, and that therefore the plate x- and y axes remain in planes parallel to the z-axis. The pitch angle θ_y describes the plate rotation about the y-axis and the roll angle φ_x describes the plate rotation about the x-axis.
Note: The subscripts in θ_y, ϕ_x refer to the axes x, y, and the angles are defined as *positive rotations* about the corresponding axes. Although this definition introduces a minus sign in the ball's equation of motion in y-direction, this definition is standard for 3D rigid transforms.

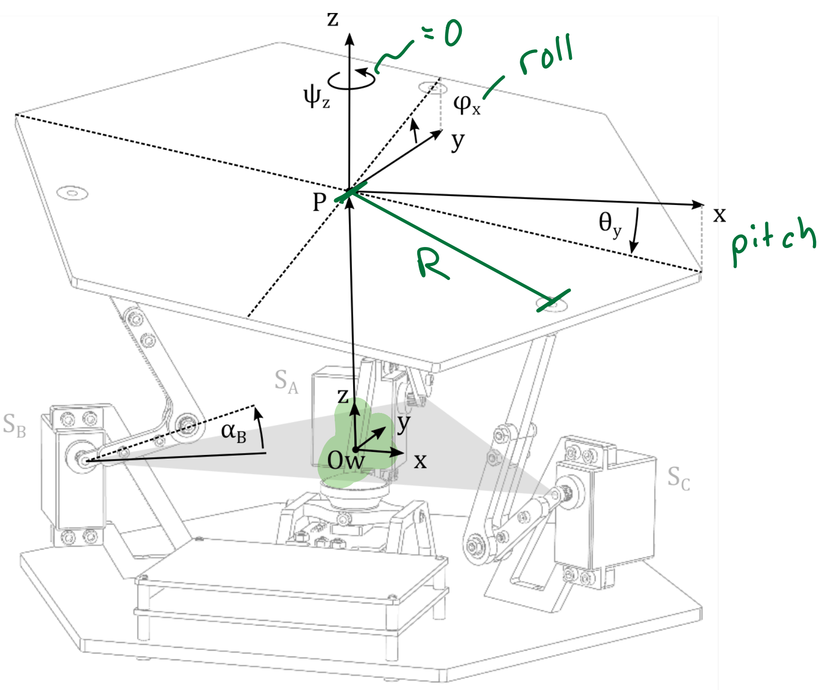


Figure 5.10: The axes in O serve as the global coordinate system. The plate center P is assumed to be at $\vec{P} = [0, 0, P_z]$, small deviations from the center are neglected. The Euler angles θ_y and φ_x describe the tilt angles of the plate and $\psi_z = 0$ is assumed. The servo angle α_B originates in the O-plane which coincides with all three servo axes. The servos are labeled S_A, S_B, S_C with corresponding servo angles $\alpha_A, \alpha_B, \alpha_C$.

Derivation

Although there is an analytic solution to the inverse kinematics problem of the 3-RRS parallel mechanism, we will derive an approximate solution, which is sufficient in precision but a lot simpler to derive and implement. To start out, we make the assumption that the angles θ_y and φ_x are independent. Consider the angle θ_y in Figure 5.11 a): To rotate the plate around the y-axis, only the joints B and C should be moved. Using trigonometry to project R onto the vertical and horizontal axes, we can compute the necessary offset in z-direction w.r.t the plate center, which is indicated as δz_A for leg A in Figure 5.11 b):

Rotation axis through actuator

$\sin(60°) = \frac{\sqrt{3}}{2}$

$\cos(30°) = \frac{\sqrt{3}}{2}$

$$\begin{aligned}\delta z_A(\theta_y) &= 0\\ \delta z_B(\theta_y) &= \frac{\sqrt{3}}{2}R\sin\theta_y\\ \delta z_C(\theta_y) &= -\frac{\sqrt{3}}{2}R\sin\theta_y\end{aligned} \tag{5.8}$$

For rotation about the x-axis, all joints have to be moved, but we can apply the same principle:

$$\begin{aligned}\delta z_A(\varphi_x) &= R\sin\varphi_x\\ \delta z_B(\varphi_x) &= -\frac{1}{2}R\sin\varphi_x\\ \delta z_C(\varphi_x) &= -\frac{1}{2}R\sin\varphi_x\end{aligned} \tag{5.9}$$

Now, using the fact that x- and y-axes are orthogonal, we add the z-offsets we obtained before to compute the final values

$$\begin{aligned}\delta z_A &= R\sin\varphi_x\\ \delta z_B &= -\frac{1}{2}R\sin\varphi_x + \frac{\sqrt{3}}{2}R\sin\theta_y\\ \delta z_C &= -\frac{1}{2}R\sin\varphi_x - \frac{\sqrt{3}}{2}R\sin\theta_y\end{aligned} \tag{5.10}$$

In the prelab, you will derive the servo angles which produce the desired height of the plate joints A, B and C using Figure 5.11.

G A G A
H H A G

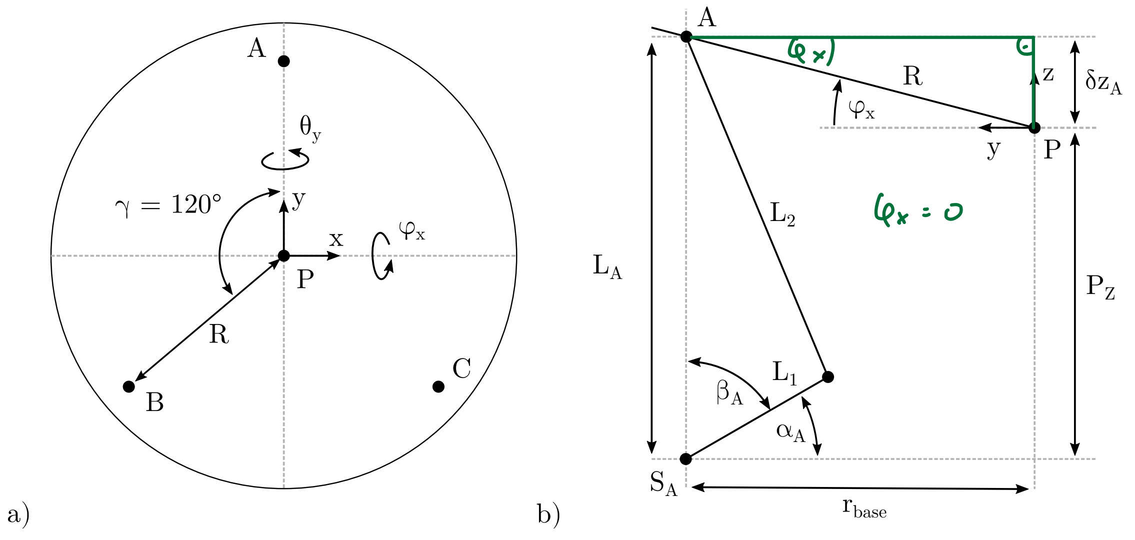
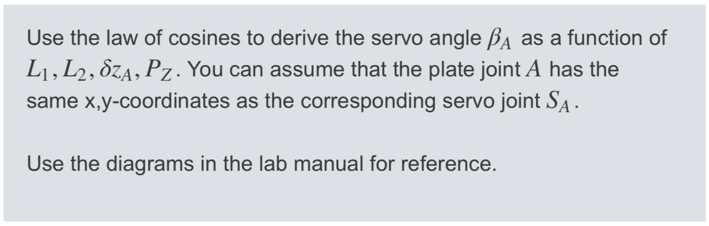


Figure 5.11: a) Top view of the plate. The plate joints A, B, C are at a distance R to the plate center P and spaced by the angle γ. Positive plate rotations are indicated by θ_y, φ_x. b) Side view of servo S_A. The spherical joint A is considered vertically above S_A. The leg consists of two links of length L_1, L_2, connected by a revolute joint. The plate center P is at a distance distance $R = \overline{AP}$ from joint A. We assume that the plate center P remains in its position at constant height P_z.


Use the law of cosines to derive the servo angle β_A as a function of $L_1, L_2, \delta z_A, P_Z$. You can assume that the plate joint A has the same x,y-coordinates as the corresponding servo joint S_A.

Use the diagrams in the lab manual for reference.

$$b^2 = a^2 + c^2 - 2ac \cdot \cos(\beta)$$

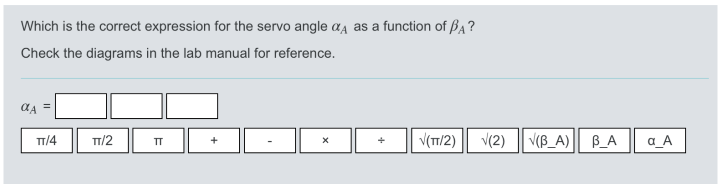
$$L_2^2 = L_1^2 + L_A^2 - 2 L_1 \cdot L_A \cdot \cos(\beta_A)$$

$$2 L_1 \cdot L_A \cos(\beta_A) = L_A^2 + L_1^2 - L_2^2$$

$$\beta_A = \arccos\left(\frac{L_A^2 + L_1^2 - L_2^2}{2 L_1 L_A}\right)$$

$$L_A = \delta z_A + P_z$$

$$\beta_A = \arccos\left(\frac{(\delta z_A + P_z)^2 + L_1^2 - L_2^2}{2 \cdot L_1 \cdot (\delta z_A + P_z)}\right)$$


Which is the correct expression for the servo angle α_A as a function of β_A?

Check the diagrams in the lab manual for reference.

α_A =

$$\frac{\pi}{2} - \beta_A$$

We want to estimate the current requirements for our servos. Using an operating voltage of **U = 7.4 V**, what is the maximum current **I** that a single servo that can draw from the power source? Check the servo datasheet for the relevant values.

- Hint 1: Assume that, in this situation, the servo exerts its maximum torque and maximum speed for a brief time interval.
- Hint 2: Unfortunately, it is common to indicate servo stall torque as **kg/cm**, although this notation does not have the unit of torque **[Nm]**. Make sure to find out how this value should be interpreted, before you start your calculations.

Calculate first the maximum power that can be generated at maximum torque and maximum speed for a brief time interval and from that calculate the maximum current.

From datasheet: $U = 7{,}4\,V \rightarrow$ Torque @ 7,4V $: 16{,}0\ ^{kg}/_{cm} \downarrow$

$1{,}569\ Nm$

$\rightarrow$ Speed @ 7,4V $= 0{,}08\ s/60°$

$= 0{,}48\ \frac{s}{360°}$

$P_{max} : T \cdot w = 1{,}569\ \frac{N}{m} \cdot 0{,}48\ ^{s}/_{360°}$

$P = U \cdot I$

$$I = \frac{1{,}569\ \frac{N}{m} \cdot 0{,}48\ ^{s}/_{360°}}{7{,}4V} =$$

5.3 Prelab Procedure

Note: The Prelab quiz on Moodle must be done before reporting to the lab. The submission deadlines are on Moodle. Late submissions will not be corrected. Each group member has to complete the quiz on their own, however you are allowed to work with students in your group to solve the quiz (and we absolutely recommend you to do so). Please remember to submit your quiz when you have answered all of the questions.

5.4 Postlab Procedure

5.4.1 Introduction

You have prepared a function to compute the inverse kinematics in the prelab. Now it is time to test it on the hardware. To set up the ball balancing system, follow the steps in the appendix. Once this is completed, you should be able to move the motors using the Arduino IDE terminal.

Attention: Do not upload the Arduino Code. This might cause damage to the servos!

Now take a look at the provided code skeleton. A few methods are already implemented, and the initialization in the main file does not need any further changes. Read through the provided code to get an overview, you will be using the same structure for the rest of the lab.
Something that might be new to you is the use of structs. We use structs to gather parameters which we want to use across the source code, such that we don't have to pass every parameter as an argument. An example of such a parameter would be the length of the first link.

5.4.2 Task 1: Inverse Kinematics

- Using the skeleton, implement the function *inverseKinematics* in `util.c` as you have prepared it in the prelab (*compute_servo_angles* function in CodeExpert). Make sure to use the parameters from the `parameters.c` file, such that you can adapt them easily later. **(Postlab Q1)**.

  Note: If you could not implement the *inverseKinematics* function, we have provided you with a precompiled library that contains the function. You can call it via the precompiled *invkin* library (see `invkin.h` for more info) and solve the tasks, but you will not receive the maximum grade.

- In `main.c` , implement a routine as follows **(Postlab Q2)**:

  1. Take user input from the terminal to specify the plate pitch and yaw angles θ_y and ϕ_x.
  2. Compute the servo angles using the *inverseKinematics* function.
  3. If the servo angles are feasible, send the command to the servos through the Featherboard.
  4. Print the Euler angles and the servo angles to the terminal.
  5. Ask the user if the process should be repeated.

- To evaluate the performance, we use the IMU installed on every modern smartphone. Install any app which displays the inclination of your phone to measure the plate angles.

  1. Set $P_z = 100\,\text{mm}$ in the `parameters.c` file and $\theta_y = \varphi_x = 0°$ manually through the terminal. Place your phone on the plate: If any angle exceeds 1°, use the servo parameters in `parameters.c` to fine-tune the corresponding servo.
  2. Use your phone to check whether the specified Euler angles are reached to satisfactory precision. The error should not exceed 2° at any angle, otherwise you should check your computations.

  Please hand in a video where you demonstrate that you can accurately command the plate angle at 0°, 5°, 10° and 15° uniaxially in both directions. The commands should be issued via terminal. We expect an error of less than 2° **(Postlab Q3)**.

  Note: Make sure the plate is mounted according to Figure 5.10

5.4.3 Task 2: Camera Calibration

- In subsection 5.2.6, the camera calibration is described. The necessary parameters are obtained using the Matlab camera calibrator, a detailed description of the process is given in the appendix. Follow the procedure carefully and obtain the following calibration parameters: distortion center, focal length, distortion parameters k_1 and k_2 as well as image size ratio (>1) between the pixy2 and your calibration images. On your report also note which system these parameters were obtained for (You can find the name next to the PCB). **(Postlab Q4)**.

  Note: It makes sense to sit at the same spot and use the same system in the lab every time, in order to not have to re-calibrate the system again. The systems in the lab are slightly different from each other.

- Implement the function *cameraCalibration* in `util.c` . This function takes in the raw image coordinates from the pixy2 and saves the calibrated coordinates in mm. Make sure to multiply the input coordinates first with the scale factor "bbs.calibration_image_scale" that you determined from your calibration procedure. This basically scales up the pixy-image such that it fits the image from the Matlab calibration procedure. Then you can proceed according to the formulas in subsection 5.2.6 **(Postlab Q5)**.

- Write code in `main.c` to test the camera calibration by using the the *cameraCalibration* function you just implemented as well as the provided *readFromPixy* function, which allows you to read the ball position from the camera. The code should do the following: If the ball is detected, its pixel coordinates and its calibrated coordinates should be visible on the terminal screen. If it is not detected, a message should appear **(Postlab Q6)**.

  Note: If you do not detect any coordinates, you have to tune the parameters of the Pixy2. Follow the instructions in the appendix to teach the camera to detect the ball.

- Using the code you just implemented in `main.c` test the camera calibration against the marks on the top plate. Have a look at Figure 5.10 to make sure the plate is mounted the right way.
  Please hand in a video with the following elements **(Postlab Q7)**:

  – Show that the coordinates in the plate center are well tuned (the calibrated coordinates should give out something close to 0 mm for x and y).

  – Show that the coordinates in the middle of the additional four engraved circles on the top plate are consistent with the ground truth (r=80 mm).

  – All demonstrations should be done at a plate height of $P_z = 100\,\text{mm}$ and $P_z = 120\,\text{mm}$ with $\theta_y = \varphi_x = 0°$. You can move the plate with the program you made for Task 1.

5.5 Postlab and lab report

1. Upload a single PDF-file with your solutions to moodle. The file should contain as all the code you implemented in the μP C files, namely submit the util.c, the main.c and your parameters.c file. Also make sure to give your obtained parameters for Postlab Q4. Please upload your solution in time (before next lab session), late submissions will not be corrected. Please also share the videos for Postlab Q3 and Q7 with your assistants on slack.

2. Come prepared with the Prelab procedure for the next lab.