

OmniWheel Robot

Leonardo David Carrera¹ Vinay Venkanagoud Patil²

¹Le7763ca-s@student.lu.se ²vi0507pa-s@student.lu.se

Abstract: In this project, a **dual-mobile robot** with a **double tracking system** is presented. It consists of a delta-based robot mounted over a 3-wheeled omnidirectional robot for fine and wide positioning, respectively. The whole system uses a **PID controller** to move the whole system, and in particular its tool center point (TCP) while it computes its position/orientation in *real-time* using the feedback from the **Lighthouse positioning system** (Base stations 2.0 from HTC Vive and a Lighthouse deck sensor from Crazyflie 2.0) and a **camera positioning system** (Raspberry camera with OpenCV computer vision library) looking downwards to the floor.

This setup aims to achieve high precision positioning (by minimizing the error between the **set goal point** and the **real goal position**) in a known environment (**testing room**) combining the accuracy of the two tracking systems while the robot performs a **coarse navigation** (omnidirectional robot) followed by a **fine positioning** (delta robot) towards the set point. Later on, we will try to optimize/maximize the goal position accuracy even more by means of: tuning the Kalman filter position estimation, selecting proper variables for both kinematic robot models, tuning PID controller parameters, and exploring several hardware and software setups.

1. Introduction

The design and control of mobile robots have been gaining importance in the technical field since their invention thanks to their multiple applications. Mobile robots are now widely used for surveillance, inspection and transportation tasks [3], for which new and improved techniques of positioning are important for studying.

Two of the most studied design challenges regarding mobile robots are accurate **positioning** and **global localization** with respect to a reference position frame. Since there are numerous applications where the navigation and localization are critical factors (often in scientific tasks) then, high maneuverability, precision positioning and smooth controllability are studied and implemented. Thus, these two challenges are assessed in this project with our setup.

In this project, the localization and positioning problems are assessed in two subtasks, where the first task is managed by the omnidirectional robot (**coarse navigation**) and the second by the delta robot (**fine positioning**).

- The first one is reaching a set position (x_{goal}, y_{goal}) given an initial position ($x_{initial}, y_{initial}$) in a global (world) coordinate reference frame $[X_w, Y_w]$ —see Figure 1—with the omnidirectional robot. Thus, we are interested in finding a trajectory with respect to this global reference frame from the initial robot pose through a certain time ($x_0(t), y_0(t), \theta_0(t)$) with the highest possible accuracy.
- The next subtask is to reach the same set position (x_{goal}, y_{goal}) with the delta robot once the omnidi-

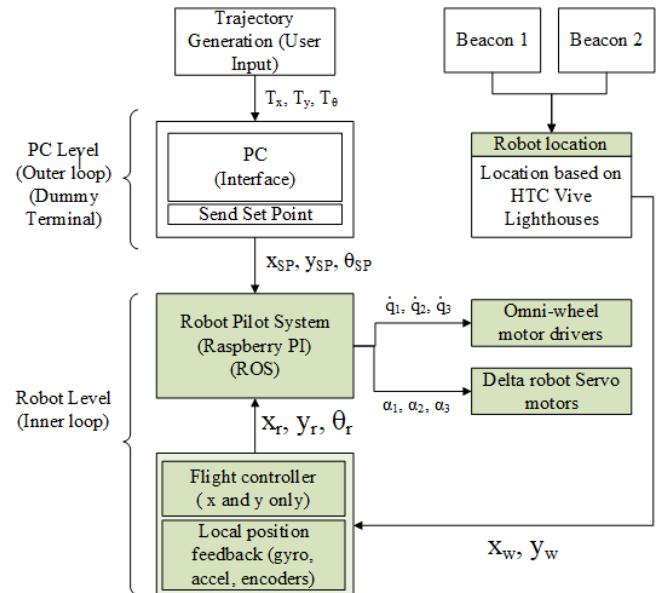


Figure 1. Flowchart of the process. The sequence ends when the error between the achieved position/orientation and the set point is small enough to consider applying a control signal. Notice that the PC interface (PC Level) is just a dummy interface for interacting with the Raspberry Pi. At this level we only perform variable inputs or monitoring (visualizing). On the Robot Level all the calculations and control are performed.

rectional robot has arrived to this desired set position. Then, the remaining distance difference (error) will be passed to the delta robot and computed as follows: $\Delta_x = x_{goal} - x_{current}$ and $\Delta_y = y_{goal} - y_{current}$ in a robot coordinate reference frame $[X_r, Y_r]$, see Fig-



Figure 2. The 3-wheeled omnidirectional robot used in this project. In this picture, neither the electronics nor the delta-frame have been installed in order to clearly show the robot frame and omniwheels.

ure 1. In other words, we will minimize Δ_x and Δ_y by translating the delta robot from the current position to the desired goal position while the omnidirectional robot is locked in that position.

A brief introduction and background theory of all the subsystems involved in the project are included as follows in order to motivate the functioning of the realized overall process, see Figure 1.

1.1 OmniWheel Robot

An omnidirectional mobile robot platform can perform translations in any direction without the need of re-orientation. This capability adds mobility to the frame and increases the robot's degree of freedom. This feature is achieved by using the resulting vectorial forces and velocities of the omnidirectional wheels which are controlled using its kinematics. Magnitudes as the speed, acceleration and strength of the motors are necessary in order to calculate the parameters of the robot model. [5]

The preferred setup for the robot platform is the **holonomic** which results from attaching omnidirectional wheels to a frame. The resulting vehicle is capable of driving along paths with independent orientation and translation. [2]

In this project, the selected setup is a 3-degree-of-freedom (vertical, horizontal and rotation) hexagonal frame which contains holders for the omni-wheels equally separated at 120° from each other. Additionally, it has 3 other mounts for the delta robot motors, see Figure 2.

1.2 Delta Robot

A Delta robot is a parallel kinematic robot with 3 degrees of freedom (x, y, z). It is known to have the ability to perform dexterous movements in the given workspace. This robot is generally used in industries [10] with requirements movements in relatively small areas. It generally is able to pick-and-place light-weighted objects with high accuracy during short cycle-times.

In this project, it is adequate to use this robot because its geometry can achieve fine positioning. It is known that

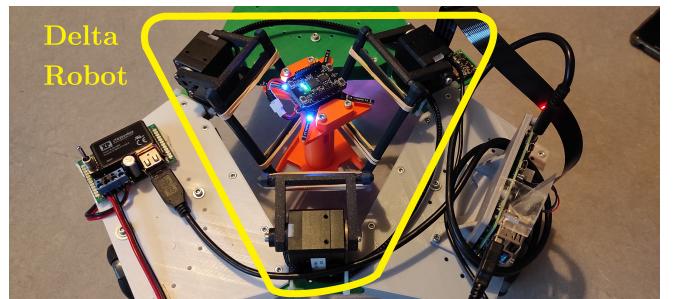


Figure 3. The Delta-mounted robot over the omni-wheeled based robot used in this project

this type of robot has serious disadvantages of a relatively limited workspace [6]. However, it is compensated with the robust and light design which maintains the repeatability of the movements. In addition, the advantage in this project is that its motors are fixed in the same rigid frame as the omnidirectional robot, reducing the shivering and lowering the center of mass of the whole robot. [10].

The Delta robot will house the flight sensor in its end effector, see Figure 3. By doing so, it is possible to track the position at all times.

1.3 The Lighthouse Positioning System

In previous works, a couple of approaches regarding mobile robots with the Lighthouse positioning system have been developed. [4, 9]. Here, we can see how good is this localization system when it comes to extract a relative position and orientation from a sensor. It is relatively low-cost (compared to other positioning methods) and can be integrated to any robotic platform with admirable results.

The lighthouse flight deck of the Crazyflie combined with the HTC-Vive/SteamVR lighthouse base stations allows a global positioning of the Crazyflie drones [1]. The data obtained from the lighthouse beacons is in essence a pair of angles describing the direction from the base station to each light sensor on the lighthouse flight deck. All measurements and position calculations are performed in the Crazyflie main board [1] which allows for introducing multiple robots in the same area without changing anything in the setup.

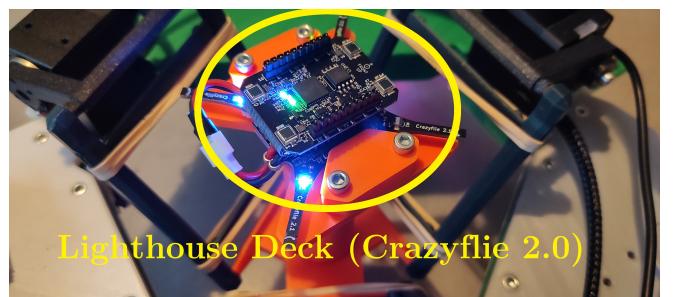


Figure 4. The Lighthouse deck over a crazyflie board. Notice the 4 photo-sensitive sensors on the Lighthouse deck

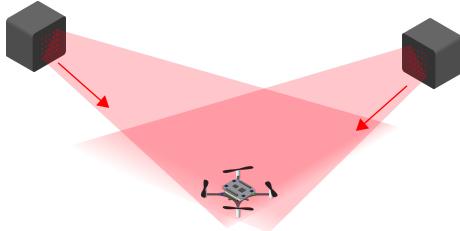


Figure 5. Crazyfly Lighthouse deck and its 4-photo-sensitive sensors detecting the sweep of the light planes emitted from the beacons (Base Stations 2.0). Taken from [1].

The crazyfly sensor (**Lighthouse deck**) —see Figure 4— captures the infrared light plane sweeps from the rotary drums in the beacons (**HTC Base Stations**) —see Figure 5— and can perform calculations over the angles of those light planes resulting in accurate position estimations (e.g. **Kalman Filter estimation**).

The Lighthouse system of Vive was originally invented/developed as a tracking system for virtual reality applications. Nevertheless, its price and robustness has made these devices very appropriate for research robotics like this one.

In this project, we are restricting the scope of the positioning system to a plane XY parallel to the floor without the height component. We will use the Kalman Filter positioning approach to tackle down the wide/fine positioning problem for the robot.

2. Modeling

In this section, we will mathematically unravel the positioning process in the different subsystems models. This will show how the navigation process is performed. The kinematics derivative in this section are based on the work in [10, 9, 3] and applied to our robots. Thus, we have,

2.1 Omnidirectional robot kinematics

Since we are using a 3-wheel omnidirectional setup —see Figure 6— we now are interested in getting the velocities of each wheel given the Cartesian velocities and the rotation velocities.

Taking R as the distance of each wheel from the center, angle α_i relative to the local frame $[x_l, y_l]$, e.g., wheel 1 is located on the local axis x_l which means, $\alpha_1 = 0$, and we can obtain the relation between the global velocity of the platform $(\dot{x}, \dot{y}, \dot{\theta})$ and the translational velocity v_i of wheel i via the inverse kinematic equation of each wheel hub i and the relationship between velocities. Thus,

$$v_i = -\sin(\theta + \alpha_i)\dot{x} + \cos(\theta + \alpha_i)\dot{y} + R\dot{\theta} \quad (1)$$

Then considering r as the radius of each wheel and the angular velocity ϕ of the wheels, then,

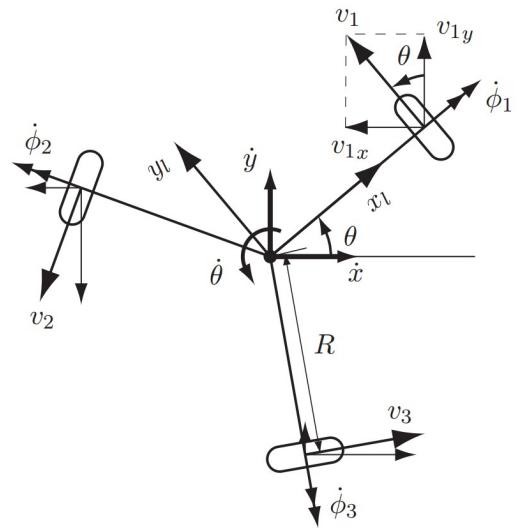


Figure 6. Kinematic diagram of the 3 wheel frame

$$\begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ \dot{\phi}_3 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} -\sin(\theta) & \cos(\theta) & R \\ -\sin(\theta + \alpha_2) & \cos(\theta + \alpha_2) & R \\ -\sin(\theta + \alpha_3) & \cos(\theta + \alpha_3) & R \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \quad (2)$$

Now, we are able to update the velocities whenever necessary in the algorithm using (2).

2.2 Delta robot kinematics

Since we are using a Delta-framed robot it is necessary for our implementation to have the inverse kinematics of this model once the omni-wheel robot has achieved its final position. Here, we are interested in getting the **three actuated revolute joint angles** if the **Cartesian position** is given.

From Figure 7 we have (3),

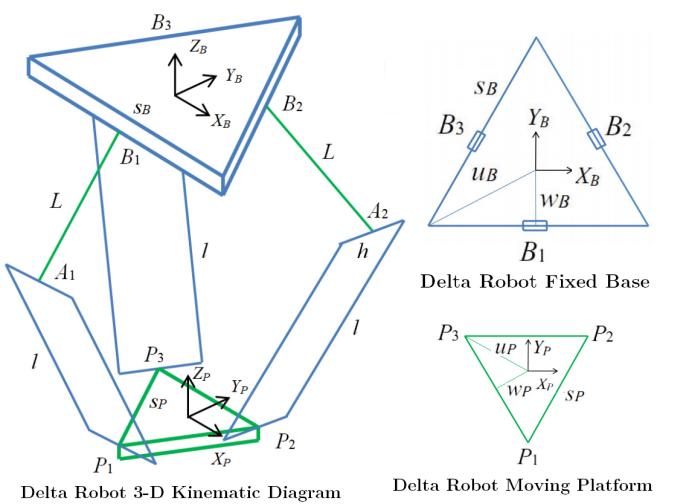


Figure 7. Kinematic diagram of Delta frame. Taken from [10]

$$\begin{aligned} a &= w_B - u_p \\ b &= \frac{s_p}{2} - \frac{\sqrt{3}}{2} w_B \\ c &= w_P - \frac{1}{2} w_B \end{aligned} \quad (3)$$

Analitically, the model is solvable if we use the three constraints equations applied to the vector loop-enclosure which are obtained from the geometry of the robot,—see Figure 7—see (4), (5), (6).

$$2L(y+a)\cos\theta_1 + 2zL\sin\theta_1 + x^2 + y^2 + z^2 + a^2 + L^2 + 2ya - l^2 = 0 \quad (4)$$

$$\begin{aligned} -L(\sqrt{3}(x+b) + y+c)\cos\theta_2 + 2zL\sin\theta_2 + x^2 + y^2 \\ + z^2 + b^2 + c^2 + L^2 + 2xb + 2yc - l^2 = 0 \end{aligned} \quad (5)$$

$$\begin{aligned} L(\sqrt{3}(x-b) - y-c)\cos\theta_3 + 2zL\sin\theta_3 + x^2 + y^2 \\ + z^2 + b^2 + c^2 + L^2 - 2xb + 2yc - l^2 = 0 \end{aligned} \quad (6)$$

We solve the inverse kinematics problem for the joint angles $\theta = \{\theta_1, \theta_2, \theta_3\}^T$ if the Cartesian position of the moving platform is given, $\mathbf{P}_P = \{x, y, z\}^T$. Additionally, and referring to the Delta Robot kinematic diagram —see Figure 7—, the IPK (Inverse Position Kinematics) problem can be solved independently for each of the three RUU legs. Thus, the three independent scalar inverse position kinematics equations are in the form:

$$E_i \cos\theta_i + F_i \sin\theta_i + G_i, \quad i = 1, 2, 3 \quad (7)$$

where E, F, G are variables which depend on the geometry equations (4), (5), (6) (since the expressions for E, F, G are extense and redundant, they are not shown in this document but one could refer to [10] to fully understand these expressions.

Thus, defining $t_i = \tan\frac{\theta_i}{2}$ and using the **Tangent Half-Angle Substitution** we have,

$$\begin{cases} \cos\theta_i = \frac{1-t_i^2}{1+t_i^2} \\ \sin\theta_i = \frac{2t_i}{1+t_i^2} \end{cases} \quad (8)$$

Substituting the **Tangent Half-Angle Substitution** (8) expressions into the EFG equation (7) we get expressions for the inverse kinematics (fully broken down in [10]) which satisfy the geometry transformation. Then, inverting the original Tangent Half-Angle Substitution definition, we have,

$$\theta_i = 2\tan^{-1}(t_i) \quad (9)$$

Two θ_i solutions result from the \pm in the quadratic formula. Both are correct since there are two valid solutions – knee left and knee right. This yields two IPK branch solutions for each leg of the Delta Robot, for a total of 8 possible valid solutions. Generally the one solution with all knees kinked out instead of in will be chosen due to physical monitoring and angle limits.

2.3 Lighthouse Positioning System (using Extended Kalman Filter)

Finally, the last stage in the navigation modeling stage is the positioning using the Extended Kalman Filter of the Lighthouse system with the Vive beacons (Base Stations 2.0) and the Crazyflie (Lighthouse Deck).

Reference frame in the Lighthouse System First, we define the reference frame used in this setup,

- Base station reference frame, see Figure 8 (a). Where the X-axis is pointing forward through the glass. The Y-axis is pointing right, when the base station is seen from the front, and the Z-axis is pointing up (away from the screw mount).
- Rotor reference frame —see Figure 8 (b)—, There is one (LH2) rotor in a **Base Station 2.0** (Beacon). Each rotor is rotating around an axis of rotation, \vec{r} . We define a coordinate system for the rotor where the Z-axis points along \vec{r} and the X-axis is the same as the X-axis of the base station.

Next, for the Base Station 2.0, we need to consider the built-in light rotor and its **Rotation Angle** —see Figure 9 (a)— and its **Light Plane Tilt**, see Figure 9 (b).

In the measurement model we want to get from a sensor position \vec{s} to rotation angle α . The first step is to calculate the sensor position in the **rotor reference frame** —see Figure 8 (b)—. The measurement is the rotation angle α when the sensor is hit by the light plane.

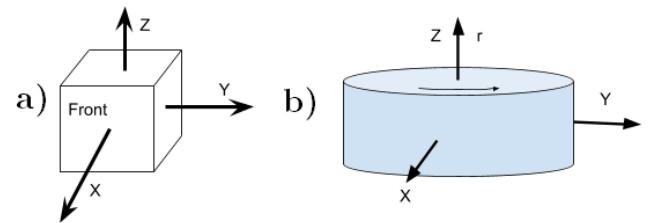


Figure 8. Reference frames used in the Lighthouse Deck. a) corresponds to the Base station reference frame and b) corresponds to the rotor reference frame. Taken from [1].

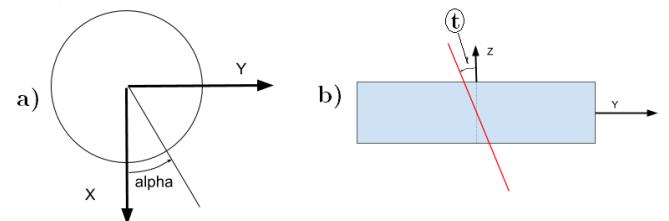


Figure 9. Components in the rotors of the Base Stations. a) Rotation Angle components. b) Light Plane Tilt components. Taken from [1].

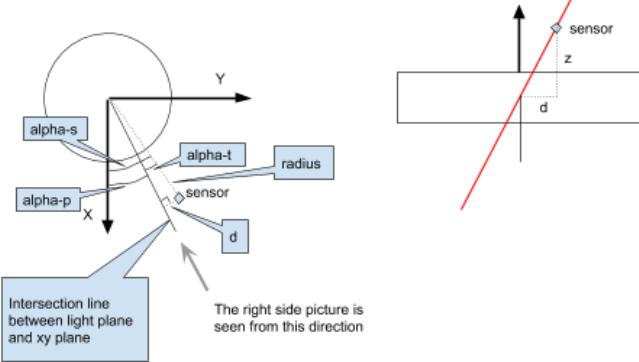


Figure 10. Rotation angles in the rotor of the Base Station. Left: Top view of the rotor. Right: Side view of the rotor. Taken from [1].

Prediction To calculate the predicted rotation angle α_p —see Figure 10—we have to go from the sensor position ($s_r = (x, y, z)$ in the rotor reference frame) to rotation angle, where the rotation angle is from the X-axis to the line where the light plane intersects the XY-plane. The rotation angle to the sensor α_s is the sum of the predicted rotation angle α_p and the rotation angle from the intersection line to the sensor α_t , caused by the tilt of the light plane, $\alpha_s = \alpha_p + \alpha_t$.

The observation model for the EKF (Extended Kalman Filter) estimates x_s, y_s, z_s components of the sensor position in the global coordinate frame, to the measured observations α_p are:

$$\alpha_p = \arctan \frac{y_s}{x_s} + \arcsin \frac{z_s \tan p}{r_s} \quad (10)$$

where,

$$r_s = \sqrt{x_s^2 + y_s^2} \quad (11)$$

The measurement model is linearized by the Jacobian:

$$\mathbf{g}_p = \left(\frac{\partial \alpha_p}{\partial x_s}, \frac{\partial \alpha_p}{\partial y_s}, \frac{\partial \alpha_p}{\partial z_s} \right) \quad (12)$$

$$\mathbf{g}_p = \left(\frac{-y_s - x_s z_s q_p}{r_s^2}, \frac{x_s - y_s z_s q_p}{r_s^2}, q_p \right) \quad (13)$$

with,

$$q_p = \frac{\tan p}{\sqrt{r_s^2 - (z_s \tan p)^2}} \quad (14)$$

This Jacobian is first rotated to the coordinate system of the base stations with the drum's rotation matrix.

With the measurement model above the EKF system on the Crazyflie is able to estimate the position for the **wide navigation** (OmniWheel robot) and the **fine navigation** (Delta Robot).

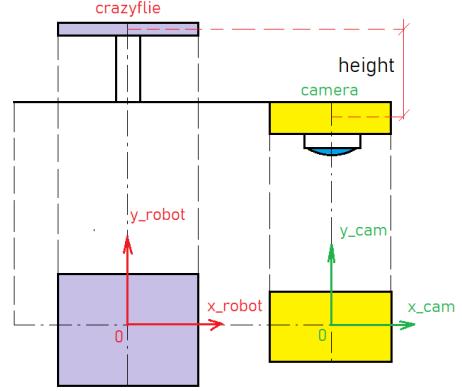


Figure 11. Relative position for the transformation matrix of camera/robot(Vive). This representation depicts in a rough way the spatial distribution of the crazyfly sensor (Lighthouse Deck) and the camera. Notice that the crazyfly is placed in the end effector of the Delta robot and the camera in the omnidirectional frame

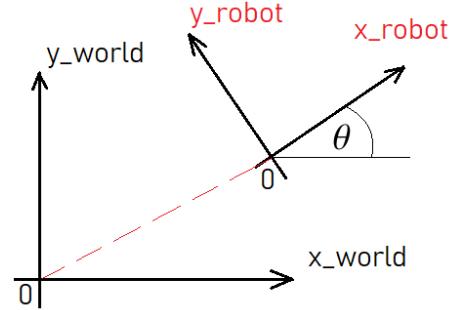


Figure 12. Relative position for the transformation matrix of robot(Vive)/world. The constitutes Delta-omnidirectional robot

2.4 Reference Positioning

Since we are dealing with two relative positions to get one global position for the robot, it is necessary to use transformation matrices from rotation followed by translation. Notice that we are only using two dimensions since the height is fixed in both cases.

The first relative position is the **camera** with respect of the **robot**—see Figure 11—for which transformation matrix will be,

$${}_{\text{robot(Vive)}} T_{\text{cam}} = \begin{pmatrix} \cos(0) & -\sin(0) & x_{rc} \\ \sin(0) & \cos(0) & y_{rc} \\ 0 & 0 & 1 \end{pmatrix} \quad (15)$$

where x_{rc} and y_{rc} in (15) represent the difference of relative position between the camera and the mobile frame of the Delta robot (mounted crazyflie).

The second relative position is the **robot** with respect of the **world**—see Figure 12—for which transformation matrix will be,

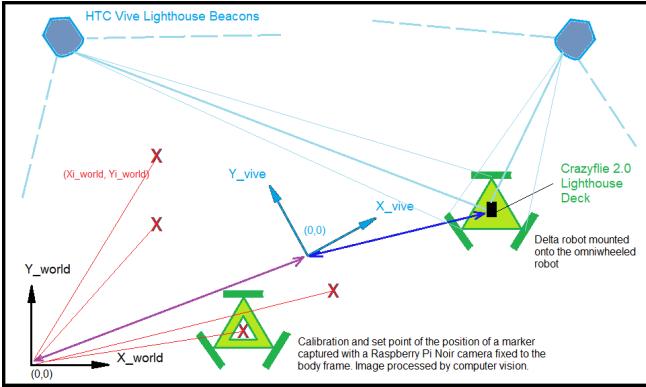


Figure 13. General architecture of the setup and how it operates. We have a broad view from the top of the setup in the testing room where the World reference frame is computed accordingly with the Robot reference frame. In addition, the red X markers are used to calibrate the two reference frames with each other.

$${}^W T_{robot(Vive)} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & x_{wr} \\ \sin(\theta) & \cos(\theta) & y_{wr} \\ 0 & 0 & 1 \end{pmatrix} \quad (16)$$

where x_{wr} and y_{wr} in (16) represent the difference of relative position between the world and the mobile omniwheeled robot (as before, from the mounted crazyflie).

3. System Design

We aim to design a system where the omni-wheel robot will be used to arrive at the set location with adequate precision. We also have a Delta robot mounted on the base which will turn be used to position the flight sensor with a much higher-precision. In the design of the control law, we have decided to move the robot in such a way that the set point is close to the center of the Delta robot's dexterous work space. The robot's omni-wheel base is a velocity based control which is much slower than the positon control of the Delta robot. Hence, the 2 levels of precision in the control of the positon of the flight sensor, see Figure 13.

3.1 Calibration

Initially, when the robot is powered on, it is required to calibrate its location within the work area. We have decided to use markers placed strategically in the area whose global positions are known. A camera is to be placed on the robot that searches for these markers and calibrates the robot's position with the help of the relative position it acquires from the Lighthouses.

The Vive Lighthouses are placed accordingly with Figure 15 and shown in Figure 16. After this, it is necessary to communicate to the crazyflie the actual position of the beacons (Lighthouses). To do that we will connect a USB cable to each beacon at a time and set the position with the installed software of crazyflie. It will upload the position of the beacons to the crazyflie itself. This procedure is executed

only once if the beacons are fixed in that position.

Once it is done it is necessary to upload the python file for the project to the crazyflie which will localize the robot in the area with respect of the Lighthouses using the EKF estimation.

After having completed the calibration sequence the robot is now ready to move to the set position. We can then provide the set location by publishing the location to a ros topic which the robot uses for set point input, see Figure 15.

3.2 Positioning

Once the relative positions are acquired, the transformation reference frame algorithm is performed with respect of each refrence frame obtaining the absolute position in the global reference frame. Thus, the robot can apply the inverse dynamics to each motor in the omnidirectional robot motor drivers and the inverse dynamics followed by the Delta robot motor drivers. The robot will perform the translation until the set point is reached. This is driven by a PI controller in the omni-directional robot. In the Delta robot a plain and simple PWM velocity control is performed.

3.3 Hardware

Since we have designed and implemented a mobile robot we have done the following tasks,

- Design and 3d-printing of the end effector of the Delta robot.
- 3d-printing of the robot parts.
- Design and implementation of the power circuit for the control board (Raspberry Pi), the camera, the Dynamixel Servos, and the electronic power coupling.
- Design and 3d-printing of the Base Stations mounts, see Figure 16.

4. Implementation

A calibration and position system is used making use of the Raspberry Pi camera mounted on the robot and OpenCV as the software library for computer vision [7]. Using the Lighthouse system and the camera positioning system the robot is able to get a global position reference of the room.



Figure 14. End effector (orange structure) and the crazyflie board with the Lighthouse deck

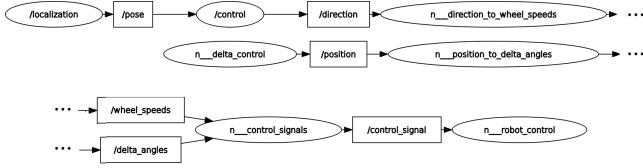


Figure 15. Nodes of the stages in the control of the OmniWheel robot and Delta robot



Figure 16. Beacons positioned in a rail with an 45 degrees angle facing downwards to map the room area in 3D

The entire source code for the robot has been divided into multiple ros nodes that function parallelly in order to move, position, and control the robot. As seen in the figure with the ros graph, there is one node on our system that supplies the robot with the setpoint over the ros network. When the robot receives this set point, it compares its current location that it acquires from the pose-estimate node, from which it computes the error and hence the correction required for it to reach the setpoint. This correction is sent to the base-control node, which computes the PWM that needs to supply to the base motors. This set of PWM values reaches the control-signal node that merges the PWM values for the base motor and also the motors in the Delta robot. This entire control signal is passed on to the robot-control node, which pushes the control signal to the respective motors, hence completing the chain of control.

5. Results

We have tested the positioning system with a number of tunable parameters. While performing the tests, we improved the accuracy of our robot controller in accordance with the whole setup (testing room and Omnidirectional robot) by **tuning the PI controller parameters**. We noticed an improvement and/or deterioration in the values of the repeatability test as the experiments were held— see Table 2 —. After numerous tests

Table 1. Values for the Omnidirectional robot PI controller.

	P (Proportional)	I (Integral)
Axis		
X	12	7
Y	12	7
θ	1.5	—

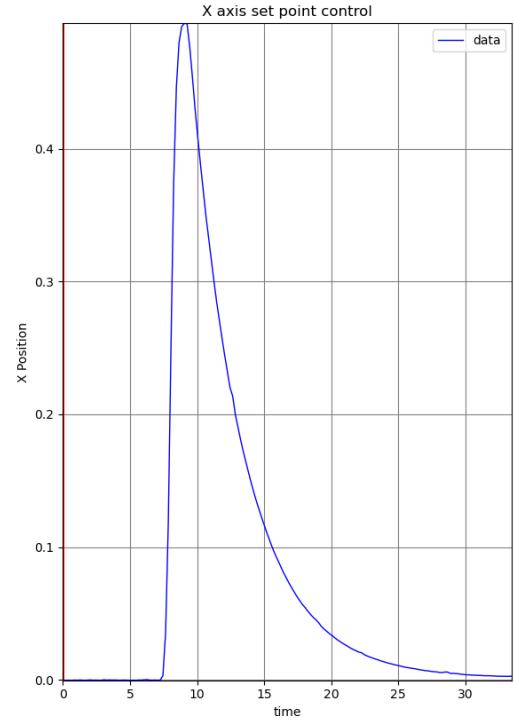


Figure 17. Position correction when displaced along x axis from its set point $x_{sp} = 0$. The horizontal axis is in seconds [s] and the vertical axis is in meters [m].

of hardware and software optimization, we set the PI values — see Table 1 — for reducing the error difference once the goal position was reached (Notice that the PI controller was implemented for the omnidirectional robot only and not in the Delta robot, which turned out it was not necessary thanks to its robust repeatability).

In order to test the the precision of the positioning (coarse and fine) of the entire robot setup, we have tested the robot in a **repeatability routine**. Accordingly, we have used a **set point** of $x_w = 0$, $y_w = 0$ and $\theta_w = 0$. Then, we *manually* have relocated the robot randomly (lifting up the robot with the hands and put the robot down in a unknown random position) within the range of the beacons (Base Stations) light planes and the sensing area of the Lighthouse Deck. Thus, we can determine how good the coarse/fine positioning are when try-

Table 2. Values with the smallest error difference in x , y , θ obtained from a series of repeatability tests when the set point was $x_w = 0$, $y_w = 0$ and $\theta_w = 0$. These values were taken after having fixed issues in the hardware setup and after have tuned the PID and Kalman estimator parameters to their best values when the error reached was the minimum possible.

Axis	Error (Omnii)	Error (Delta)
	[mm]	[mm]
Δ_x	3.649	2.141
Δ_y	4.817	1.682
Δ_θ	2.035	1.942

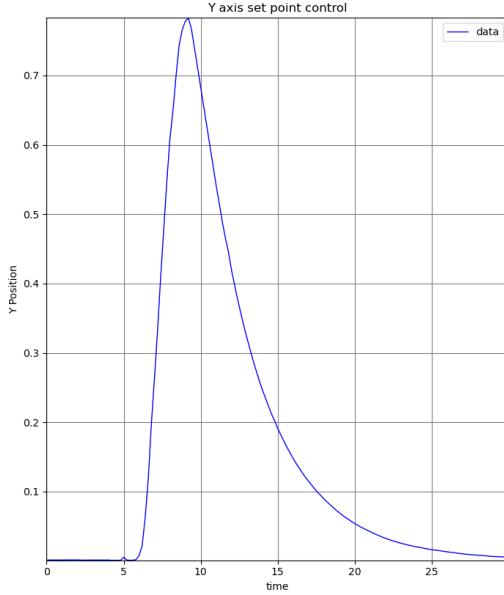


Figure 18. Position correction when displaced along y axis from its set point $y_{sp} = 0$. The horizontal axis is in seconds [s] and the vertical axis is in meters [m].

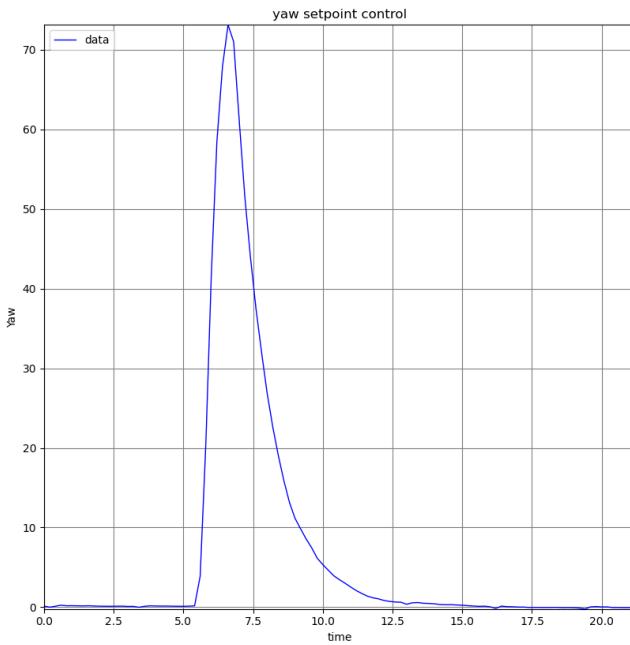


Figure 19. Rotation correction when rotated around z axis from its set point (yaw angle) $\theta_{sp} = 0$. The horizontal axis is in seconds [s] and the vertical axis is in degrees [$^\circ$].

ing to return to the *zero* set position. Also, we can determine hardware issues just by looking at the error values in each axis and angle rotation, see Figures 17, 18, 19, 20.

6. Discussion

We have investigated and implemented the coarse navigation and the fine positioning of this Omniwheel robot with two low-cost precise-tracking systems in an attempt to reduce the

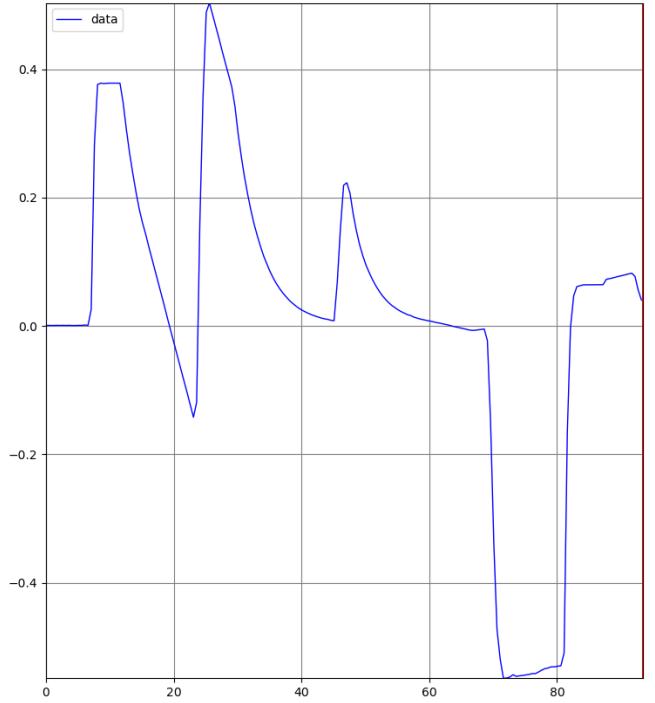


Figure 20. Repeatability test along x axis. We have performed a random pick and place of the robot expecting from the robot to return to its zero position along the x-axis. The horizontal axis is in seconds [s] and the vertical axis is in meters [m].

inherent error of such tasks. We have noticed how sensitive the hardware is when it comes to apply models specially designed for them. Since the models are idealized structures (algorithms and mathematical models), the error will always be present in the positioning/localization tasks.

The redundancy of the Lighthouse system, which in fact is remarkable, has a range or error values which are impossible to estimate below the millimeters level. Nevertheless, considering the area covered by the robot ($3.5[\text{m}] \times 3.5[\text{m}]$ approximately), the positioning accuracy in [mm] is uncanny.

Thus, one can realize that the use of the Delta robot is justified if one tries to obtain extra precision by positioning the system as precisely as possible.

If one would try to reduce this range of millimeters to a few microns, the work would then consist in applying more precise systems such as computer vision for absolute positioning, laser tracking and/or mapping/radar sensors such as Lidar. Likewise, given that this project used custom-made parts with a certain range of error in manufacturing, the next challenge would be to improve the manufacture of these parts and achieve a model very similar to that obtained from mathematical models and simulations.

With this project we have been able to develop a clear and brief vision of how the global location problem of a mobile robot could be addressed for high precision positioning

applications. One of the most important points is the use of specialized operating systems for robots such as ROS [8], which allows the integration of almost any type of system, algorithm, sensor, electronics, mathematical model and programming languages in a single environment. This allows to the designer to perform complicated tasks in a structured way. In this project, for example, the use of ROS allowed adapting the positioning systems with the inverse kinematics of the respective robots in a harmonious way.

References

- [1] Bitcraze. *The lighthouse positioning system*. 2021. URL: <https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/lighthouse/> (visited on 2021).
- [2] M. D. Correia, A. Gustavo, and S. Conceição. “Modeling of a three wheeled omnidirectional robot including friction models”. *IFAC Proceedings Volumes* **45**:22 (2012). 10th IFAC Symposium on Robot Control, pp. 7–12. ISSN: 1474-6670. doi: <https://doi.org/10.3182/20120905-3-HR-2030.00002>. URL: <https://www.sciencedirect.com/science/article/pii/S1474667016335807>.
- [3] G. I. R. K. Galgamuwa, L. K. G. Liyanage, M. P. B. Ekanayake, and B. G. L. T. Samaranayake. “Simplified controller for three wheeled omni directional mobile robot” (2015), pp. 314–319. doi: [10.1109/ICIINFS.2015.7399030](https://doi.org/10.1109/ICIINFS.2015.7399030).
- [4] M. Greiff, A. Robertsson, and K. Berntorp. “Performance bounds in positioning with the vive lighthouse system”. English. In: *International Conference on Information Fusion (FUSION)*. 22th International Conference on Information Fusion (FUSION) ; Conference date: 02-07-2019 Through 05-07-2019. 2019. URL: <https://www.fusion2019.org/>.
- [5] R. Haendel. “Design of an omnidirectional universal mobile platform”. *Journal of Educational Psychology - J EDUC PSYCHOL* (2005).
- [6] X.-J. Liu, J. Wang, k. oh ku, and J. Kim. “A new approach to the design of a delta robot with a desired workspace”. *Journal of Intelligent and Robotic Systems* **39** (2004), pp. 209–225. doi: [10.1023/B:JINT.0000015403.67717.68](https://doi.org/10.1023/B:JINT.0000015403.67717.68).
- [7] P. Orgeira-Crespo, C. Ulloa, G. Rey-Gonzalez, and J. A. Pérez García. “Methodology for indoor positioning and landing of an unmanned aerial vehicle in a smart manufacturing plant for light part delivery”. *Electronics* **9**:10 (2020). ISSN: 2079-9292. doi: [10.3390/electronics9101680](https://doi.org/10.3390/electronics9101680). URL: <https://www.mdpi.com/2079-9292/9/10/1680>.
- [8] Stanford Artificial Intelligence Laboratory et al. *Robotic operating system*. Version ROS Melodic Morenia. 23, 2018. URL: <https://www.ros.org>.
- [9] A. Taffanel, B. Rousselot, J. Danielsson, K. McGuire, K. Richardsson, M. Eliasson, T. Antonsson, and W. Hönig. “Lighthouse positioning system: dataset, accuracy, and precision for UAV research”. *CoRR abs/2104.11523* (2021). arXiv: [2104.11523](https://arxiv.org/abs/2104.11523). URL: <https://arxiv.org/abs/2104.11523>.
- [10] R. L. Williams. “The delta parallel robot: kinematics solutions”. *Mechanical Engineering, Ohio University* (January 2016). URL: www.ohio.edu/people/williar4/html/pdf/DeltaKin.pdf.

LUND UNIVERSITY
AUTOMATIC CONTROL
FRTN70 PROJECT IN SYSTEMS, CONTROL AND LEARNING
SPRING 2021

OmniWheel Robot

Project Plan Group J

Leonardo David Carrera¹ Vinay Venkanagoud Patil²

Project Advisor: Anders Robertsson

¹le7763ca-s@student.lu.se
²vi0507pa-s@student.lu.se

1 Project Purpose

The goal of this project is to model the forward and inverse velocity kinematics of a 3-OmniWheel mobile robot and the inverse kinematics model for small motions of a Delta robot mounted in the mobile robot frame. In addition, reach an accurate robot position (using the robots combined) from the global position in room (by using the HTC Vive Lighthouse tracking system) with respect of its local position (using a quadcopter flight controller).

2 Equipments and material

In this section we propose a list containing the materials needed for the project. We have appended the approximate costs of each item on the list — based on the research we have been doing online.

2.1 Robotics

- Metal frame. (approx. 1000 SEK)
- Dynamixel MX-64T Servo motors x 3 (one per axle of the mobile frame). (approx. 2000 SEK/each)
- Dynamixel MX-64T Servo motors x 3 (one per arm of the Delta robot). (approx. 2000 SEK/each)
- Raspberry Pi 4 Model B - 4GB. (approx. 690 SEK)
- Crazyflie-Flow v2 deck (flight controller) for local positioning. (approx. 450 SEK)
- 12 - 18 VDC power supply for servos, flight controller and Raspberry. (approx. 800 SEK)
- Omni wheels x 3. (approx. 110 SEK/each)

2.2 Global positioning system

- HTC Vive Cosmos. (approx. 8990 SEK)
- HTC VIVE Base Station 1.0 (Lighthouses). (approx. 1037.57 SEK)

3 Modelling and System Design

For this project, we have considered several stages described in a flowchart (Figure 1). Each component can be developed and implemented independently from one another in early stages. Once the stages have been sufficiently completed, they will be integrated to function as a single positioning unit for the mobile robot.

The hardware setup will include:

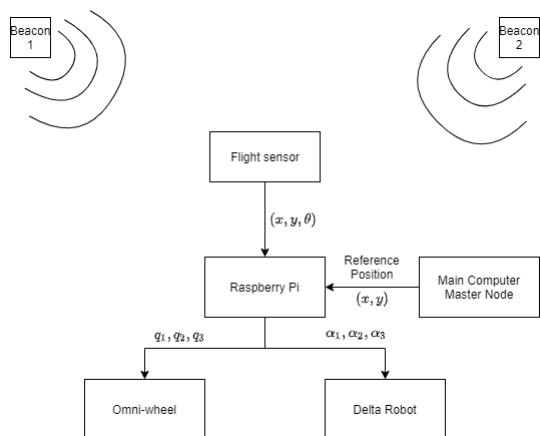


Figure 1: Software Architecture.

- The Delta/Omni-wheeled robot.
- Two beacons (HTC lighthouses) mounted in the testing room that will be used for the precise localization of the robot.
- An Raspberry Pi to control the robot and process the localization data..
- A PC to run The HTC Vive software and set point the starting position.

For the implementation we propose to use a software structure seen in Figure 1. We have decided to use a ROS framework to control the Robot and to communicate with an external PC which will be used to set the setpoint for the robot and also to collect data for analysis. The Robot will be equipped with a flight sensor which will be obtaining the localization data using the help of the lighthouse beacons.

4 Division of labour

We have decided to split our project into three parts: Omniwheel modelling of vectors (forward/inverse velocity kinematics), positioning in the room (globally), Delta robot inverse kinematics for small motion/fine tuning positioning (Optional).

4.1 Omniwheel modelling of vectors

This is the first stage of our project where we will develop a specific kinematic model for the three-omni-wheeled robot according to its dimensions. After we have obtained the kinematic model of the robot, we will proceed to implement it in the ROS environment installed in the robot-mounted Raspberry Pi taking into account the local (flight controller) positioning system.

Both of us will be in charge of this section. Since this task is complex and demanding (and also the main one) we have decided to implement it both. We will take turns developing the mathematical model of the robot and implementing the code in ROS.

4.2 Positioning in room (global positioning)

Once the model is functional for the local positioning of the robot on the horizontal / vertical axis plus its orientation, we will use the data obtained from the robot to compare it with the global positioning system (HTC Lighthouses) and accurately move the robot to a previously given location.

Leonardo Carrera will be in charge of the HTC Lighthouse positioning system in the main computer and Vinay Patil will be in charge of the calculations between the beacon position data (HTC Lighthouses) and the flight controller sensor.

4.3 Delta robot inverse kinematics for small motion/fine tuning positioning (Optional)

After having reached an approximate position from the beacons of the HTC Lighthouse and the flight controller, the next task is to implement fine positioning with small movements generated by a Delta robot mounted on the frame of the mobile robot. Thus, a reflector mounted on the tool center point of the delta robot will have the ability to move finely and reach the specified position with millimeter precision.

As before, the task is moderately complex, so we will both design the mathematical positioning model and the corresponding programming in ROS.

5 Time Plan

5.1 Subtasks

- Configure a ROS environment with all the necessary APIs and libraries needed and test with all the robot components. **Estimated deadline:** 31/3

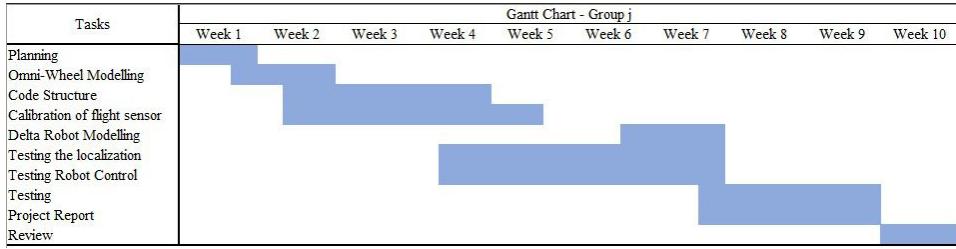


Figure 2: The Gantt chart describing the work flow of our project.

- 3d-print custom parts for holding motors and sensors. **Estimated deadline:** 7/4
- Assemble the omni-wheel and Delta robot. **Estimated deadline:** 10/4
- Set up the beacons of the HTC Lighthouse system in stationary positions and set up the area for the robot. **Estimated deadline:** 14/4
- Implement the communication between Raspberry and PC (ROS messages, ROS Wrapper, etc). **Estimated deadline:** 21/4
- Find mathematical models for both, the omni-wheel kinematic model and the fine tuning delta-robot kinematic model. **Estimated deadline:** 29/4
- Model the algorithm for getting the local position with respect of the global position. **Estimated deadline:** 29/4
- Create a software package for the Dynamixel servos in ROS. **Estimated deadline:** 6/5
- Implement the mathematical models in ROS. **Estimated deadline:** 14/5
- Tune the models with the actual robot and correct positioning faults (hardware and software). **Estimated deadline:** 15/5

5.2 Important dates

- Mar 29 - Hand in project plan.
- Apr 22 - Feedback seminar 1 on the modeling and design.
- May 5 - Report should be pushed to git to allow peer review by other groups.
- May 11 - Peer review done
- May 12 - Feedback seminar 2 on the design and implementation.
- May 20 - Project done and demonstrated and final report handed in
- May 27 - Demo film upload and peer review of final report done
- June 3 - Final presentation and demonstration and revised final report handed in

5.3 Gantt Chart

We have formalized the time plan as a Gantt chart. The major tasks can be seen plotted in Figure 2.