Real-time vision-based controller for delta robots

Article *in* International Journal of Intelligent Systems Technologies and Applications · January 2021

DOI: 10.1504/IJISTA.2021.121321

2 authors:

Ali Sharida
Texas A&M University
11 PUBLICATIONS **10** CITATIONS
SEE PROFILE

Iyad Hashlamon
Palestine Polytechnic University
20 PUBLICATIONS **126** CITATIONS
SEE PROFILE

Real-time vision-based controller for delta robots

Ali Sharida* and Iyad Hashlamon

Mechanical Engineering Department,
Palestine Polytechnic University,
Hebron, Palestine
Email: 156049@ppu.edu.ps
Email: iyad@ppu.edu
*Corresponding author

Abstract: This paper investigates two real-time vision-based control algorithms for delta robots. The first one aims to enable the robot to track different objects based on their colours and shapes. This algorithm does not need any initial calibration. Instead, it depends on the least squares algorithm (LSA) to generate the required transformation matrixes. Also, it is implemented on a standalone controller with no additional time complexity added to the main controller. The second one is a self-calibrating human hand gesture tracking algorithm, which can perform automatic calibration and generates transformation matrixes automatically based on the initial measurements of the user's body. The algorithms are designed, implemented, and scheduled in a real-time manner. The results show that these algorithms can track fast-moving objects effectively regardless of the initial configuration of the robot. They provide important solutions for common problems related to visual servoing such as field of view and calibration.

Keywords: vision-based control; real-time control; delta robot; visual servoing; hand gestures tracking.

Reference to this paper should be made as follows: Sharida, A. and Hashlamon, I. (2021) 'Real-time vision-based controller for delta robots', *Int. J. Intelligent Systems Technologies and Applications*, Vol. 20, No. 4, pp.271–295.

Biographical notes: Ali Sharida received his BE in Mechatronics Engineering from Palestine Technical University (PTUK), Tulkarm, Palestine in 2013, and MSc in Mechatronics Engineering from Palestine Polytechnic University (PPU), Hebron, Palestine in 2020. In 2014, he joined PTUK TA. His current research interests include robotics modelling and design, motion control, intelligent systems, and systems identification. This research is a part of the Master thesis in PPU.

Iyad Hashlamon received his BE in Mechatronics Engineering from Palestine Polytechnic University (PPU), Hebron, Palestine in 2004, MSc and PhD in Mechatronics Engineering from Sabanci University, Istanbul, Turkey in 2010 and 2014, respectively. In 2014, he joined PPU as an Assistant Professor. His current research interests include robotics modelling and design, motion control, intelligent systems, humanoid robots and systems identification.

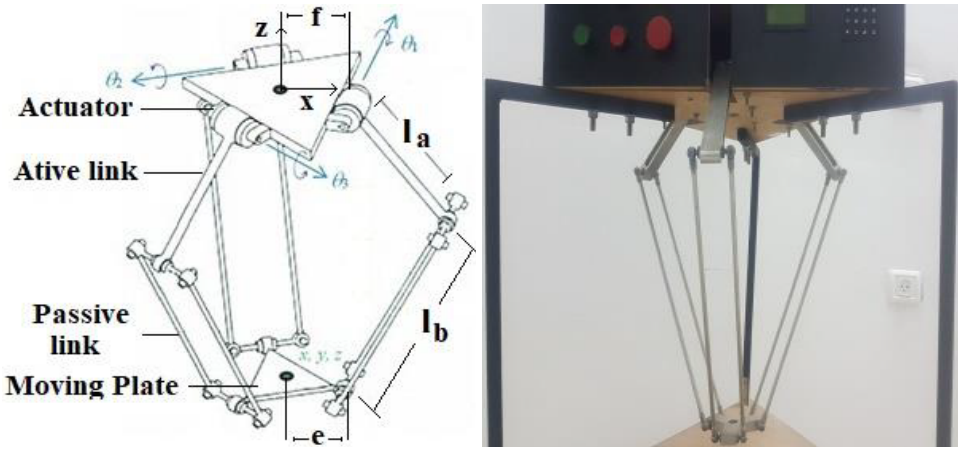
1 Introduction

Delta robot is a three degree of freedom (DOF) parallel robot, it consists of three closed kinematic chains, each chain is composed of one active link and one passive link, the former is connected to the fixed plate while the latter is connected to the active link. The moving platform is attached to the end of the passive links as shown in Figure1.a. The hardware prototype is shown in Figure 1(b).

This mechanical structure provides multiple advantages for delta robots to be used in industrial applications which need speed, accuracy, repeatability, and force distribution (Li et al., 2019). For these properties, delta robots are widely used in high-speed applications such as picking and placing (Brinker et al., 2017), high accuracy applications (McClintock et al., 2018), assembly tasks (Bulej et al., 2018), CNC machines (Correa et al., 2016), 3D printing (Rehman et al., 2019), haptic controllers (Mitsantisuk et al., 2015) and multiple other applications (Singh et al., 2013).

Vision-based control (visual servoing) has been widely spread in industrial applications such as picking and placing (López-Nicolás et al., 2019) to increase production efficiency. It is a technique that depends on the data acquired from a visual sensor (for example a camera) to track a specific object and to control the motion of a robot accordingly.

Figure 1 (a) Robot kinematic diagram (b) Robot hardware prototype (see online version for colours)



(a) (b)

Vision-based control techniques attracted many researchers for better interaction between robots and the surroundings to increase the efficiency, accuracy, and speed of the robots. Some of the researchers developed algorithms of visual servoing to estimate the position, speed, and orientation of the end effector as in Sahu et al. (2019), Fang (2011) and Ficocelli and Janabi-Sharifi (2001), they depend on visual feedback closed-loop control to perform direct vision-based control. This method is very efficient for slow robots. However, their efficiency is decreased dramatically as the speed of the robot is increased due to the constraints of the camera frame speed and the high computational time required to process each frame.

Others used the visual servoing to track specific objects in industrial applications especially for picking and placing as Gridseth et al. (2016), Rouhollahi et al. (2018), Li et al. (2017) and Hu et al. (2016), while other applications employed it to perform contour tracking for drilling, printing and welding functions as in Chang et al. (2017) and Li et al. (2018). However, these studies have a major drawback as they require an initial calibration, which requires a high knowledge and experience of the calibration process. A small error during calibration may cause an incremental error during operating the system.

Visual servoing is also used frequently in obstacles detection and avoidance as Asadi et al. (2019), Hafez et al. (2017) and Shi et al. (2018), the researchers depend on a visual method to detect the obstacles located on the robot's way and the robot should avoid them. The major disadvantage of this method is that it cannot guarantee that the target object is in the field of view all the time.

This paper overcomes the aforementioned drawbacks. The camera is located at the centre of the robot frame, so the task space is completely located in the field of view all the time. More, the initial calibration is not required, since the algorithms depend on least squares algorithm (LSA) and human body measurements to generate the required transformation matrixes regardless of the initial position or configuration of the robot. To overcome the computational time related to image processing and its effects on sampling time, all tasks related to recognition and tracking were implemented on a stand-alone device, which means that no additional computational time will be added to the computation of the control law on the main controller.

The main contribution of this paper is developing two real-time vision-based control algorithms. Both of them guarantee efficient tracking of fast-moving objects, overcome the problem of the field of view and the initial calibration and finally they guarantee a minimised computational time. The proposed algorithms depend mainly on a stand-alone controller to minimise computational time and a distributed control network to achieve the control tasks in a real-time manner.

The rest of the paper is organised as follows, the problem statement is introduced in Section 2. Section 3 presents the description of the system. Object recognition and tracking are described in Section 4. Hand gestures recognition and tracking are introduced in Section 5. The trajectory is designed in Section 6. The real-time scheduling is shown in Section 7, and the overall system is discussed in Section 8. Results are shown in Section 9. Finally, the paper is concluded in Section 10.

2 Problem statement

Industrial processes require effective and interactive feedback about the position and the speed of the objects in the workspace. This feedback increases the efficiency, speed, and accuracy of the industrial process. Vision-based control is used for this purpose, where visual feedback can be obtained by utilising a visual sensor.

The visual sensor acquires data about the configuration of the robot (direct visual servoing) or the tracked objects (indirect visual servoing). The acquired data is then supplied to the controller as a reference signal or feedback. However, the known algorithms – that are used for visual servoing – contain difficulties such as the field of view, the time complexity, the necessity of calibration, and the effects of the surrounding

environment on the efficiency of recognition and tracking. Moreover, in practical applications timing constraints are very critical and may deteriorate the response of the robot.

These problems should be solved in a real-time manner to obtain a minimised sampling time which ensures the stability of the robot and the accuracy of performing its functions.

This paper investigates two vision feedback algorithms (VFA) to provide the control law with the required variables related to the position and the velocity of the tracked objects, to provide a visual reference signal. The proposed algorithms can track the objects based on their shapes or colours without the need for calibration knowledge. All required transformation matrixes are generated automatically using LSA. More, an additional application was proposed to perform self-calibration hand gesture tracking using Kinect, which can be used in multiple applications such as manual picking and placing and surgery.

3 System description

This paper is based on previous work Sharida and Hashlamon (2020, 2019) where a real-time distributed controller was designed to control the delta robot. The real-time network is designed based on controller area network (CAN) bus protocol. It uses four microcontroller units (MCU), each one consists of a microcontroller and a CAN bus receiver-transmitter. One MCU is used to compute the control law while the other three MCU's are connected to the actuated joints through an electronic interfacing module. Each one of the three MCU's along with the actuator and sensory system forms an intelligent sensor-actuator-system (ISAS).

Each ISAS can communicate with other ISAS's and the controller MCU through CAN bus communication protocol. The ISAS reads the actuator position through an encoder, forms the necessary signal processing and prepares the ready measured data in a massage then broadcasts it to the CAN bus. This message will be received by the beneficiary MCU, and in the same way for all ISASs.

The controller MCU computes the required control law and broadcasts it on the CAN bus. Each ISAS will receive its massage and skip the others. Then each ISAS analyses the message and applies the required signal to the actuator. This approach enhances flexibility to the system for changing the control approach and adding other jobs in addition to distributing the computational load among four MCUs. To apply the vision-based controller, an additional node will be added to the system to provide the reference signal to the controller. All recognition and tracking tasks will be implemented on this node using an Android smart device where no additional load will be added to the main controller.

The results of this node are shared with the main controller as packets of data using Wi-Fi depending on user datagram protocol (UDP) using an Android smartphone. This provides multiple advantages over other image processing devices. On one hand, it is supplied with a built-in camera, this feature reduces the processing time as there is no need to use any hardware connections to transmit the image to a standalone processor. On the other hand, modern smartphones have high-resolution cameras and high-speed processors. A standalone camera with high resolution cannot be used without a computer

and requires additional installations and drivers to become compatible with industrial projects.

The device will capture frames of images to track the specified object continuously. As a frame becomes ready, the tracking cycle is triggered. Frame capturing rate is adjusted on the maximum frame rate supported by the device which is 60 frames per second (FPS). This means that the tracking algorithm's frequency (TF) is 60 Hz.

4 Object recognition and tracking

Object recognition is responsible to extract data from input images. It consists of four steps:

1 Receiving an input frame and analysing it to obtain the position of the tracked object (object recognition).

2 Transforming the position from the camera frame to the robot frame.

3 Computing the linear velocity of the tracked object by taking the time derivative of its position.

4 Sending the position and the velocity of the object continuously to the main controller as a reference signal which should be tracked by the controller (object tracking). In this paper, the recognition and tracking algorithm aims to recognise objects based on their shapes or colours. Then, it should extract the position of the target object in the camera frame (pixel units). This algorithm was implemented using OpenCV library (Bradski and Kaehler, 2008) for image processing for Android operating systems (OS). The selection of such a library provides multiple advantages as Android OS is open-source and its application can be easily implemented. Furthermore, smartphones have internal cameras and internal communication methods such as Wi-Fi and Bluetooth, so data can be shared easily after processing. Accordingly, objects will be detected and tracked using the developed Android application.

The obtained position of the object is transformed using the robot's inverse kinematics into the joint space to provide the reference signal. Finally, the reference signal is transmitted to the main hardware controller using Wi-Fi by the mean of UDP protocol.

4.1 Object recognition

4.1.1 Object recognition for circular objects

This option enables the robot to track only circular objects neglecting their colours. This algorithm depends on Hough circle transform (HCT) (Supriyanti et al., 2012) which finds circles in a greyscale image. It defines the circle with a centre point (consists of X and Y locations) and a radius. The central location can be used to get the position of the object in X and Y-axes, while the radius will be used to compute the position on Z-axis.

At the first step, the camera is initialised and started to obtain images continuously, these images are converted from RGB to the greyscale. As the algorithm is responsible to track the circular objects regarding their colours, the greyscale is entirely sufficient for

this task, so there is no need to use a complicated process for coloured images which requires more computational time. The greyed images are then blurred to remove the outlier pixels that represent noise in the image. The data relating to the position of the tracked object can be obtained by applying the HCT algorithm to the blurred images. If the algorithm finds multiple circles, the circle with the largest radius will be selected for tracking, and its parameters will be printed and stored for the next processes and control. As shown in Figure 2, the steps of this algorithm can be summarised as,

1 Initialising the camera by setting the frame size (800 × 600) and the capturing speed (60 FPS).

2 Receiving the new frame.

3 Converting the frame to greyscale.

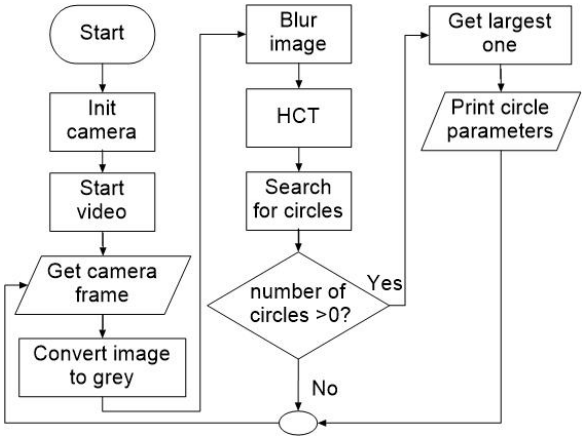
4 The frame is blurred to reduce the noise.

5 Applying the HCT algorithm on the blurred image to obtain the circles.

6 If the number of the found circles is larger than 0, the algorithm will print the parameters of the largest circle, in other words, the circle that has the largest radius.

Figures 3 and 4 show the results of this algorithm, where the frame (image) contains three objects: large red circle, small red circle, and blue rectangular object. In Figure 3, the algorithm detected the large circle as was expected, and it neglects the small one. The large one was removed in Figure 4, so the algorithm detects the small one.

Figure 2 Circular objects recognition algorithm



4.1.2 Object recognition for coloured objects

This option enables the robot to recognise the objects depending on their colours. The algorithm is started by selecting the desired colour by the user and initialising the camera. When a new image is received, each pixel of the image is converted to its equivalent RGB value to obtain the values of red, green, and blue ratios. These ratios are used to compare if this pixel matches the desired colour specified by the user. If the obtained ratios are in the range of the target colour, the algorithm stores the position of the

matched pixel. When all pixels are analysed, the algorithm computes the centre point of the object by computing the average location of the matching pixels. The average location is then used to obtain the position of the tracked object in the XY plane, while the number of the matching pixels is used to determine the position of the tracked object in the Z-axis.

Figure 3 Large circle detection (see online version for colours)

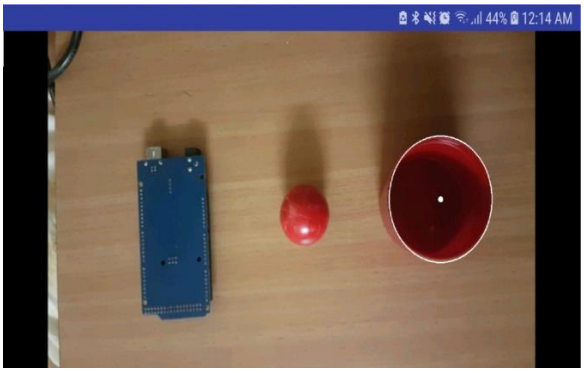
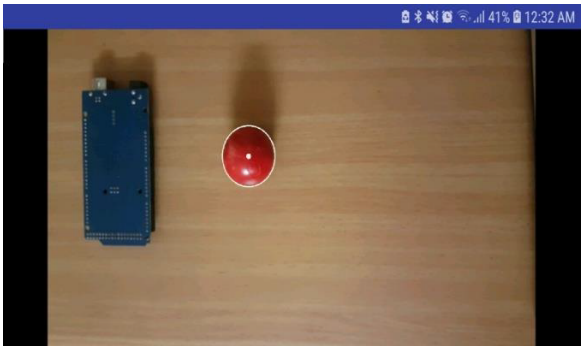


Figure 4 Small circle detection (see online version for colours)



As shown in Figure 5, the steps of this algorithm can be summarised as,

1 Initialising the camera by setting the frame size (800 × 600) and the capturing speed (60 FPS).

2 Receiving the new frame.

3 It checks the colour of each point in the image.

4 If the colour of this point matches the target colour, the point location will be stored in a list until the end of the cycle.

5 All matched points are used to compute the average (centre) location of the object on the XY plane using the following relation:

$$x = \frac{\sum_{i=0}^{p} x_i}{p}, \; y = \frac{\sum_{i=0}^{p} y_i}{p} \tag{1}$$

where x, y are the central location of the detected object in the camera frame and x_i, y_i are the location of the i^{th} pixel, p is the number of the matched points.

6 The number of the matched points is then used to compute the position of the target object on the Z- axis.

Figure 5 Object recognition for coloured objects

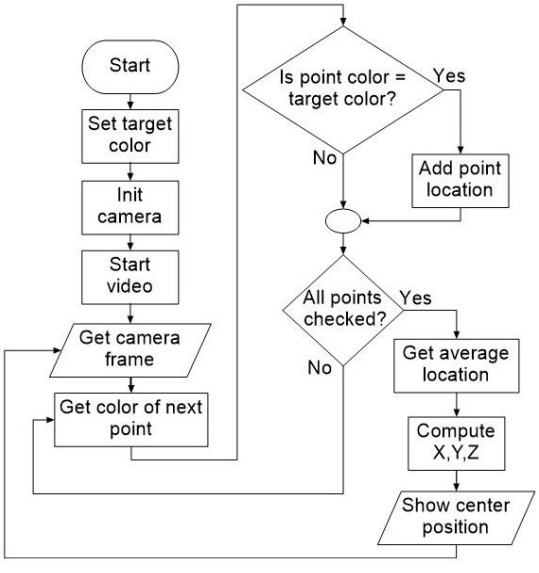


Figure 6 shows the detection resulted from this algorithm for selecting blue objects. The image shows a frame with multiple objects, one of them is blue (Arduino Mega at the left) and the algorithm detects it.

Figure 6 Colour detection (see online version for colours)

4.2 Position transformation from camera frame to robot frame

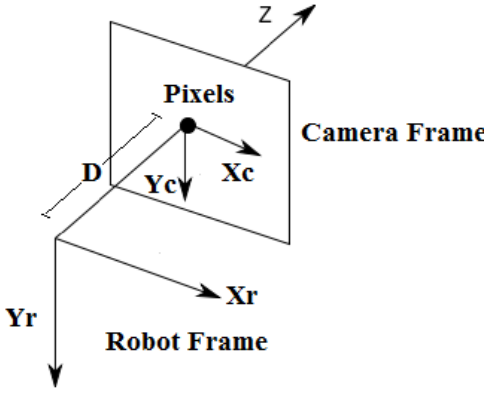
Usually, the transformation from the camera frame to the robot frame is carried out using two steps (Andhare and Rawat, 2016):

1 Transformation of pixels to cm in the camera frame.

2 Transformation of cm in the camera frame to the robot frame.

These steps require a relation between the position obtained in pixels and the real position related to the camera frame. The results obtained from this relation are then transformed to the robot frame using a transformation matrix. Figure 7 shows the relation between the camera frame and the robot frame.

Figure 7 Position transformations



The offset between the camera frame and the robot frame can be represented by a linear distance D and rotational offset δ ([D δ]) with small constant values (Agudo et al., 2016). However, underestimating them will cause tracking errors. This error occurs due to imperfect matching between the camera frame and robot frame during camera setup. To eliminate this error, both (D) and (δ) should be estimated.

This paper proposes an empirical method to estimate and derive a transformation matrix that transforms local camera frame (pixel) to global robot frame (cm) with eliminating the effect of the offset among frames. This method provides the advantage of reducing computational time and compensates the errors resulted from the imperfect positioning of the camera at the origin of the robot frame. The main challenge after object recognition is how to transform the local position (pixel) to the global position (cm). Converting pixels to cm is a linear process (Agudo et al., 2016), which depends on two variables, pixels on X-axis (X_c) and pixels on Y-axis (Y_c). This can be estimated using the LSA depending on training data. From practical observation, it was noticed that the function of the real position on X-axis (X_r) and Y-axis (Y_r) is linear of two independent variables (X_c and Y_c) measured by pixels in the camera frame, while the real position on Z-axis (Z_r) is a function of a single variable (Z_c) in the camera frame. These relations are described in equation (2) and will be proven experimentally in this section.

$$\begin{aligned} X_r(X_c, Y_c) &= a_x + b_x X_c + c_x Y_c \\ Y_r(X_c, Y_c) &= a_y + b_y X_c + c_y Y_c \\ Z_r(Z_c) &= a_z + b_z Z_c \end{aligned} \tag{2}$$

where a_x, b_x, c_x, a_y, b_y, c_y, z_z and b_z are constants to be estimated, X_c, Y_c and Z_c are the position of the tracked object in camera frame measured in pixels. Then, the global transformation matrix can be written depending on equation (2) as follows,

$$\begin{bmatrix} X_r \\ Y_r \\ Z_r \end{bmatrix} = \begin{bmatrix} b_x & c_x & 0 \\ b_y & c_y & 0 \\ 0 & 0 & b_z \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} + \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$$

where all parameters are scalars. Each sub-equation in equation (2) will be analysed separately to obtain the unknown parameters. For the training data, the first part of equation (2) is rewritten into matrix form as a function of the measured and unknown parameters as:

$$X_r = A \begin{bmatrix} a_x \\ b_x \\ c_x \end{bmatrix} \tag{3}$$

where

$$A = \begin{bmatrix} 1 & 1 & 1 & . & . & 1 \\ X_{c1} & X_{c2} & X_{c3} & . & . & X_{ci} \\ Y_{c1} & Y_{c2} & Y_{c3} & . & . & c_i \end{bmatrix}^T \tag{4}$$

$$X_r = \begin{bmatrix} X_{r1} & X_{r2} & X_{r3} & \dots & X_{ri} \end{bmatrix}^T, \tag{5}$$

And i is the number of experiments. The previous equation can be solved using the following formula of the LSA (Haddad et al., 2019):

$$Q \begin{bmatrix} a_x \\ b_x \\ c_x \end{bmatrix} = A^T X_r \tag{6}$$

where

$$Q = A^T A \tag{7}$$

$$A = \begin{bmatrix} N & \Sigma X_{ci} & \Sigma Y_{ci} \\ \Sigma X_{ci} & \Sigma X_{ci}^2 & \Sigma X_{ci} Y_{ci} \\ \Sigma Y_{ci} & \Sigma X_{ci} Y_{ci} & \Sigma Y_{ci}^2 \end{bmatrix} \tag{8}$$

$$A^T X_r = \begin{bmatrix} \Sigma X_r & \Sigma X_r X_{ci} & \Sigma X_r Y_{ci} \end{bmatrix}^T \tag{9}$$

Finally, the vector of unknown parameters can be written as follows,

$$\begin{bmatrix} a_x \\ b_x \\ c_x \end{bmatrix} = Q^{-1}\left(A^T X_r\right) \tag{10}$$

This LSA is solved automatically at the starting of the software using practical training data. In the beginning, equation (10) was solved to find the relation of X_r as a function of X_c and Y_c obtained from the camera, then the same procedure is applied to obtain a relation of Y_r and Z_r. The results were obtained as three vectors as follows:

$$\Phi_x = \begin{bmatrix} a_x \\ b_x \\ c_x \end{bmatrix} = \begin{bmatrix} -4.2233 \\ 0.0198 \\ -0.0264 \end{bmatrix}, \Phi_y = \begin{bmatrix} a_y \\ b_y \\ c_y \end{bmatrix} = \begin{bmatrix} 24.6718 \\ -0.0247 \\ -0.0204 \end{bmatrix}, \Phi_z = \begin{bmatrix} a_z \\ b_z \end{bmatrix} = \begin{bmatrix} 12.26 \\ -0.0325 \end{bmatrix}$$

The resulted fitting curves between the real position (X_r) and the estimated position (X_e) are shown in Figures 8 and 9 for X and Y-axes respectively. The mean square error was computed for each curve as 0.3 cm and 0.27 cm respectively. They confirm the assumption that the relation between the camera frame and the robot frame is linear as was assumed in equation (2).

4.3 Linear velocity extraction

In vision-based control, it is important to determine the position and the velocity of the moving object. Both of them will be used as reference signals for the controller. The position of the moving object represents the destination of the robot's endpoint, while the velocity of the moving object represents the desired velocity of the endpoint. To extract the velocity of the tracked object, the time derivative of the object's position is considered, this value represents the instant velocity of the object and can be computed as follows:

$$V = \frac{X_{e_2} - X_{e_1}}{t_s} \tag{11}$$

where V is the velocity of the tracked object, X_{e_2} and X_{e_1} are the estimated positions obtained from two sequential frames, and t_s is sampling time for frames capturing.

Figure 8 X-axis curve fitting (see online version for colours)

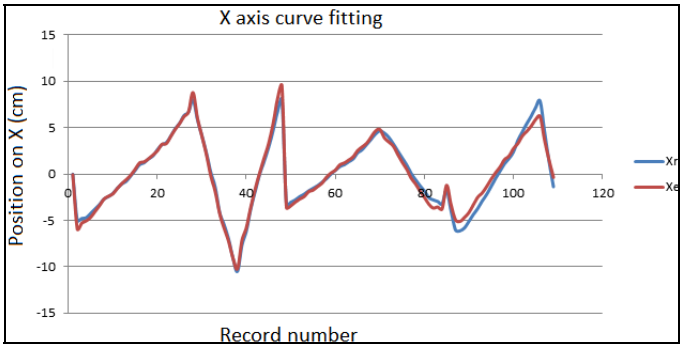
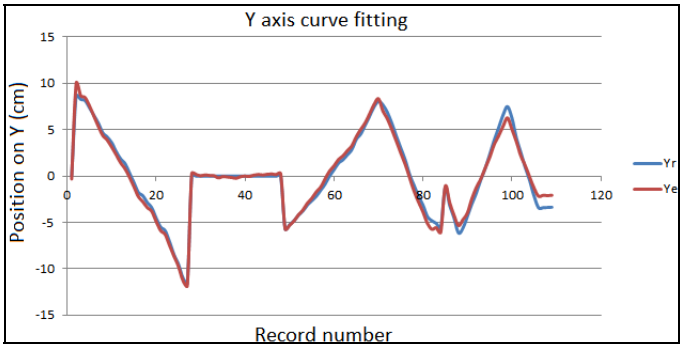


Figure 9 Y-axis curve fitting (see online version for colours)



4.4 Object tracking

Object tracking is performed by transmitting the packets that contain the position and velocity of the tracked object continuously. If any value of the position or velocity is changed, the smartphone will create a packet of tracking data and send it to the main controller, which will, in turn, generates the control signal that is required to track the received reference signal.

This operation is continuously repeated as the object is moving. When its velocity becomes 0, the tracking algorithm will transmit a packet to the controller to inform it that the speed of the object is zero, so the controller creates a new trajectory such that the final position of the trajectory equals the reference position (position of the tracked object) and the final velocity is zero.

5 Hand gestures recognition and tracking

Gestures recognition and tracking are very useful in robotics applications such as manual picking and placing, medical applications (Wachs et al., 2011; Becker et al., 2013), and body motion simulation (Gans et al., 2009; Mills et al., 2011). It aims to provide the robot with the ability to track the user's hand through two dimensions (X, Z) as the user faces the XZ plane of the camera.

The Kinect camera was used to achieve this goal due to its main advantage of the internal software development kit (SDK) supported by MICROSOFT's programming languages. This SDK provides the programmer with the advantage of obtaining the position of the joints in the XZ plan easily. While the depth (Y) can be obtained by the attached infrared sensor. So, the programmer does not need to recognise the joints of the human body, instead, he should deal with the obtained coordinates of each human joint. When the Kinect recognises a human body, it computes the depth distance of this body using the attached infrared sensor. Then, its SDK provides details about the position of each joint in that body. In fact, not all joints are important for hand tracking tasks. Therefore, for simplicity, only the important joints will be imported from the SDK as shown in Figure 10.

At the first step, the algorithm creates a local (xz_l) and a global coordinate (XZ_g) in the camera frame. The global coordinates lay on the top right corner of the computer screen,

while the local coordinate was selected to be located at the centrw point of the right shoulder as shown in Figure 11. The SDK provides the programmer all joints coordinates with respect to the global coordinate.

Accordingly, transformation procedures will be done at each program cycle. In this application, the transformation process between these coordinates is linear, as the local coordinate is linearly transformed by a distance D from the global coordinate, this distance can be computed using the global coordinate of the right shoulder (X_s, Z_s) by the following relation,

$$D = \sqrt{X_s^2 + Z_s^2} \tag{12}$$

At each program cycle, the algorithm focuses on the local coordinate, where its origin is the global coordinates of the right shoulder. So, all other joints will be transformed from the global coordinates to the local coordinates, hence, the location of each joint should be known with respect to the right shoulder (X_s, Z_s).

Figure 10 Human joints (see online version for colours)

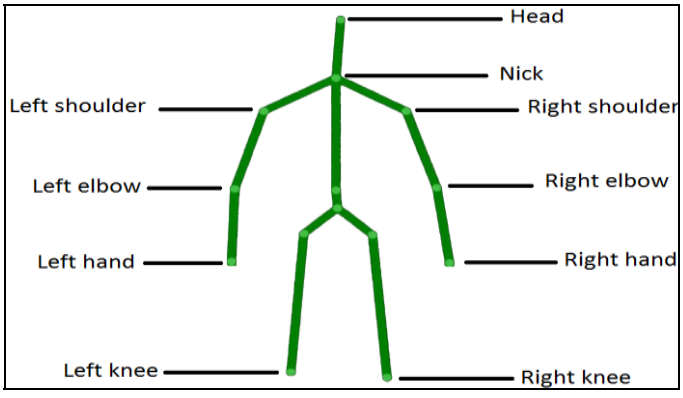
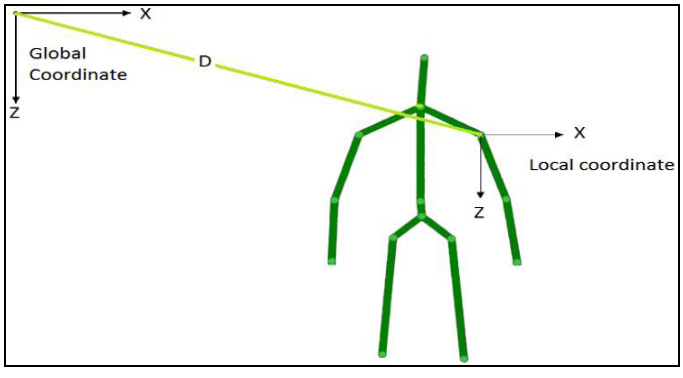


Figure 11 Global and local camera coordinates (see online version for colours)



Any transformation is executed by subtracting the global position of the right shoulder from the global position of the desired human joint as the following,

$$x_j = X_j - X_s$$
$$z_j = Z_j - Z_s \tag{13}$$

where x_j and z_j represent the local coordinate of the joint, X_j, Z_j are the global coordinate of the joint. As the position of the right shoulder is the centre of the local coordinate (xz_l), its local coordinate is always (0, 0). And using equation (13), the local coordinates of other joints can be computed.

The next step is to transform the local coordinates to the robot coordinates. This is accomplished by creating a transformation matrix, which has constant elements and can be obtained using the equation of the straight line. This process represents a linear transformation as the user can move the endpoint of the robot by his hand gestures, for example, as the user raises his hand upward, the endpoint of the robot goes up for a distance linearly dependent on the user arm displacement, and vice versa.

To apply this, the local coordinate should be calibrated and the length of the right arm should be measured. The proposed hand gesture tracking algorithm provides the user the ability to calibrate all parameters by standing on location facing the camera, then by raising the left hand to a level above the left shoulder as shown in Figure 12, the system will auto-calibrate all required parameters. As the user performs this action, the position of the right shoulder will be recorded for future transformations. The length of the right arm (L) can be computed by the following relation,

$$L = \sqrt{x_{rh}^2 + z_{rh}^2} \tag{14}$$

where (x_{rh}, z_{rh}) is the local coordinate of the right hand.

The human hand can be rotated in a circular motion to control the position of the robot's endpoint, with a radius of the arm length (L). Moving the hand from the maximum local x position $(x_{H_{\max}})$ to the minimum local x position $(x_{H_{\min}})$ will move the robot from maximum x robot position $(x_{R_{\max}})$ to the minimum x robot position $(x_{R_{\min}})$. And the same is for the motion on the z-axis.

To obtain the equivalent position required for the moving plate, the equation of the straight line $y - y_1 = S(x - x_1)$ will be employed as follows,

$$x_R - x_{R_{\max}} = S\left(x_H - x_{H_{\max}}\right) \tag{15}$$

where S is the slope of the line, $S = \dfrac{x_{R_{\max}} - x_{R_{\min}}}{x_{H_{\max}} - x_{H_{\min}}}$.

From Figure 12, $x_{H_{\max}}$ is ($+L$) and $x_{H_{\min}}$ is ($-L$), so the equation (15) becomes,

$$x_R = \frac{x_{R_{\max}} - x_{R_{\min}}}{2L}\left(x_H - x_{H_{\max}}\right) + x_{R_{\max}} \tag{16}$$

and the same procedures can be done to obtain transformation for the robot z coordinate.

$$z_R = \frac{z_{R_{\max}} - z_{R_{\min}}}{2L}\left(z_H - z_{H_{\max}}\right) + z_{R_{\max}} \tag{17}$$

Finally, the obtained data is transformed from task space to joint space and sent as a socket reference signal to the control MCU using a USB serial port. These steps are described in the following pseudo-code and the flow chart in Figure 13.

Figure 12 Start calibration gesture (see online version for colours)

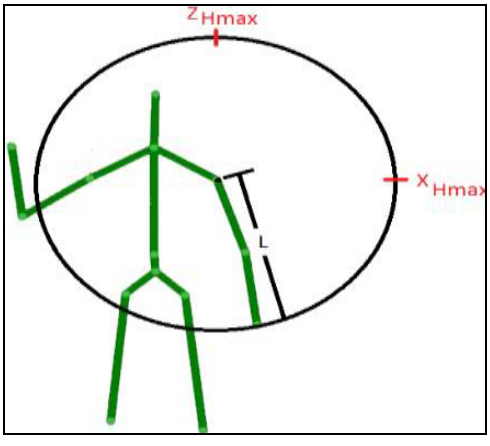
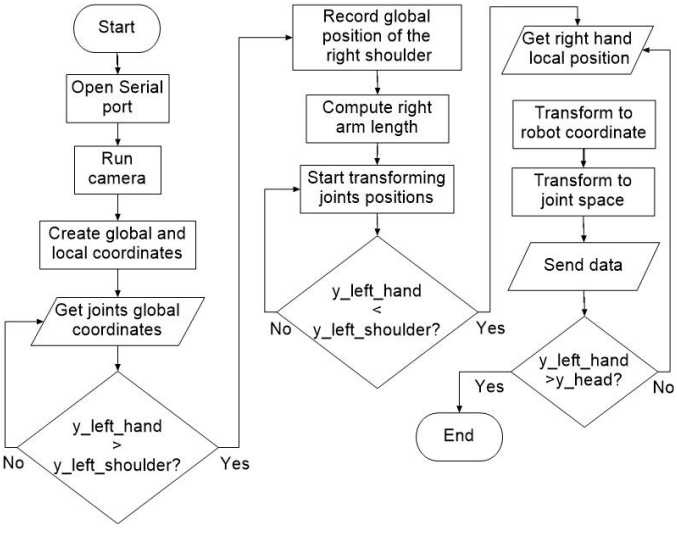


Figure 13 Hand gesture tracking algorithm

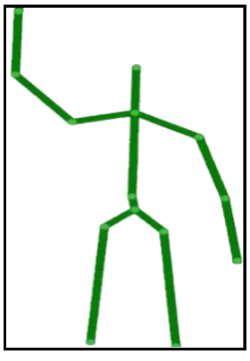


Pseudo code:

- Open the serial port.
- Run the camera.
- Create the global and local coordinates.
- Wait until the user raises his left arm above the level of his left shoulder to calibrate, then:
  - a Record the global position of the right shoulder.
  - b Compute the right arm length.
  - c Start transforming joint's positions from the global to the local coordinates.

- Wait until the user lowers his left hand.
- Get the local position of the right hand.
- Transform the right hand's local position to the robot coordinates.
- Transform to joints space using inverse kinematics.
- Create and send the reference signal as a socket.
- End the tracking if the user raises his left hand above the head's level, as shown in Figure 14.

Figure 14 End gesture tracking (see online version for colours)



6 Trajectory design

Trajectory planning is used to provide a smooth reference signal depending on the estimated position of the tracked object, this signal will ensure that the moving platform will move in the desired manner with a soft start and stop. The trajectory should provide the signal that translates the moving platform from the initial position X_0 to the final position X_1 during a period (t). Furthermore, to ensure a smooth motion, the velocity of the end effector at the initial time (t_0) and final time (t_1) should be 0:

$$\begin{aligned} &X(t_0) = X_0,\ X(t_1) = X_1 \\ &\dot{X}(t_0) = 0,\ \dot{X}(t_1) = 0 \end{aligned} \tag{18}$$

In equation (18) there are four constraints that the trajectory should satisfy, therefore, the equation of the moving platform's position with respect to time must be a cubic polynomial as

$$X_T(t) = a_0 t^3 + a_1 t^2 + a_2 t + a_3 \tag{19}$$

Using the constraints provided in equation (18), a system of four equations can be created, this system was solved with the following results:

$$
\begin{aligned}
a_0 &= \frac{2(X_0 - X_1) + (\dot{X}_0 - \dot{X}_1)(t_1 - t_0)}{(t_1 - t_0)^3} \\
a_1 &= \frac{3(X_1 - X_0) - (\dot{X}_1 + 2\dot{X}_0)(t_1 - t_0)}{(t_1 - t_0)^4} \\
a_2 &= \dot{X}_0 \\
a_3 &= X_0
\end{aligned}
\tag{20}
$$

7 Real-time scheduling and schedulability test

To ensure real-time tracking, tasks scheduling and schedulability tests should be analysed.

Capturing frame rate was adjusted to 60 Hz, which means a frame will be ready each 160 ms. In the previous project (Sharida and Hashlamon, 2020), the control sampling time was selected to be 8.5 ms. As control sampling frequency is more than ten times larger than tracking frequency, the system is stable from the sampling time point of view.

Detection and tracking consist of multiple tasks, computing the position of the object relative to the camera frame, transforming the position to the robot frame, inverse kinematics, and transmitting data as a reference signal using Wi-Fi. Table 1 summarises these tasks and their specifications.

The first task is to receive a new frame from the camera, it was defined as a sporadic task because all next processes will depend on the results obtained from this frame. So, when a new frame is ready, the processor will terminate any under execution task and will not complete the uncompleted job, further, the processor will reset all periodic tasks and execute them again. Therefore, these tasks should be executed before receiving a new frame which represents a deadline for the remaining tasks.

Table 1 Tasks specifications

Task	*Name*	*Type*	*Execution time (ms)*
1	Receive new frame	Sporadic	5.2
2	Object detection (circular)	periodic	4
3	Object detection (colour)	Periodic	5.8
4	Position transformation	Periodic	0.1
5	Inverse kinematics	Periodic	0.2
6	Send data over Wi-Fi	Periodic	1.2

The next task is to detect the tracked object, which can be done using two user-defined methods. The user can select one of them at the beginning of the software without considering the selection time in the total computational time. For shape-based detection, it requires 4 ms in the worst case, where no circles are found. So, the worst-case execution time will be assumed as 4 ms. For colour-based tracking, it needs a fixed time of 5.8 ms to test all points and extract their colours to get the canter of the object.

The next one is to extract position data from the obtained pixels position, in other words, this task represents the transformation from camera frame to robot frame. This

task can be done in either the Android application or in the main controller. However, the Android processor has more idle time than the main controller. Therefore, it's preferred to execute it with the next task in the Android processor instead of executing them on the main controller.

The obtained real position represents an input for the next step, where inverse kinematics is used to transform the position from workspace to joints space. In this task, the angular position is obtained to form the reference signal in the control algorithm. This task requires 0.2 ms to be executed using the developed application. Finally, transmitting data over Wi-Fi requires 1.2 ms to be executed, this task contains multiple jobs including network communications and authentications, and this time is required in the worst case when a new Wi-Fi connection is required.

When the camera frame is ready to be executed, all other tasks are triggered. Receiving a frame from the camera is a sporadic task. After receiving this frame, the application will find the position of the target object using one of the aforementioned algorithms. When this task is finished, it triggers the next task to transform the position from camera frame to robot frame.

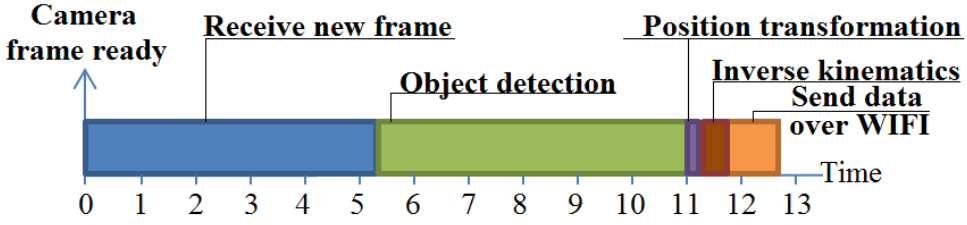
The estimated position in the task space is transformed to joint space using the robot's inverse kinematic. Finally, the joint angular position is sent over Wi-Fi using the UDP protocol. To ensure that all packets had been received successfully, the transmitter will get a replay from the receiver which contains the reference angular position. If the replay contains disturbing data, the transmitter will send the packet again and will skip it if the data is correct. Furthermore, if the packet is lost, the receiver will not replay before the deadline of replaying timeout (1 ms), so that, the transmitter will send the same packet again.

From this description, it can be realised that all tasks are dependent and none of them can be pre-empted by another periodic task. However, receiving a new frame which is a sporadic task can pre-empt any periodic task at any time when it is released.

Worst-case execution time (WCET) occurs when the user selects to track an object based on its colour and a new image frame is released. This case requires executing all tasks described in Table 1. So, the processor needs 12.5 ms to achieve all required tasks. All specifications were measured experimentally by getting operating system time (Android) after each task.

Since the required tracking sampling time is larger than the WCET, the system is schedulable logically. The rate monotonic (RM) utilisation test ensures that the system is schedulable too, as the utilisation factor is less than 10%which keeps it idle for more than 90% of its cycle time as shown in Figure 15.

Figure 15 Tracking tasks scheduling (see online version for colours)



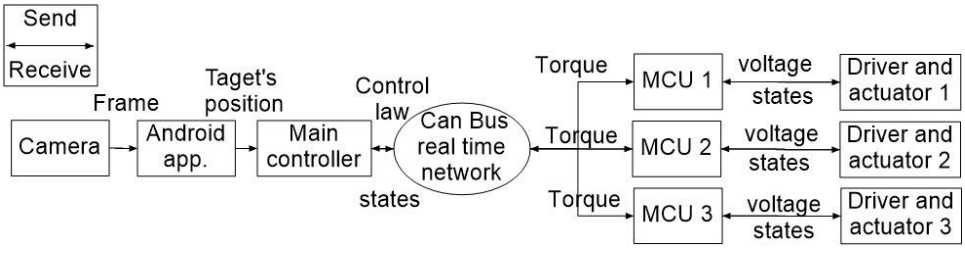
8 Overall vision-based controller

The overall setup for this project is shown in Figure 16. The Android device is fixed at the origin of the robot axes. However, it has some deflection in the orientation, as it is very difficult to match the position and orientation between the two spaces. As a frame is ready, the Android application will analyse this frame to get the estimated position of the tracked object. The estimated position is then transmitted to the main controller which in turn computes the control law and shares its value over the CAN bus real-time network. Each ISAS receives the control law and applies it to the actuator through the electrical driver. Finally, each ISAS sends the states of the attached actuator to the main controller to find the error and perform a closed-loop control.

9 Results

The proposed algorithms were implemented and tested experimentally on the delta robot shown in Figure 1 which is a lab-made robot that was designed and developed in the PPU university. For the objects tracking algorithm, a sample ball was tracked. Figure 17 shows the ball before starting the tracking process. When the ball is in the reachable task space, the tracking process starts automatically, and the moving platform of the robot will keep tracking it until the ball is at the centre of the moving platform as shown in Figures 18 and 19. For continuous object motion, the robot will keep tracking the object until its velocity is zero, Figures 20 and 21, show the results of continuous tracking of the ball with random motion on the x- and y-axis respectively. In industrial applications especially in picking and placing, the objects had a constant z location as they are usually moved by a conveyer belt or located at the ground, for this reason, the z-axis for the object was neglected in this experiment.

Figure 16 Overall system block diagram



The hand gesture tracking algorithm was also tested for random hand motion. At the start-up of the algorithm, it searches for a human body, once it is found, the algorithm locates the global and local coordinates as shown in Figure 22. When the user raises his left hand, the calibration and transformation matrix will be generated automatically and the tracking process is triggered. Then the moving platform will continuously move proportionally to the motion of the hand as shown in Figures 23, 24, and 25.

Figure 17 Object detection at the boundaries of task space (see online version for colours)

Figure 18 Object tracking first example (see online version for colours)

Figure 19 Object tracking, the second example (see online version for colours)

Figure 20 Object motion tracking on the x-axis (see online version for colours)

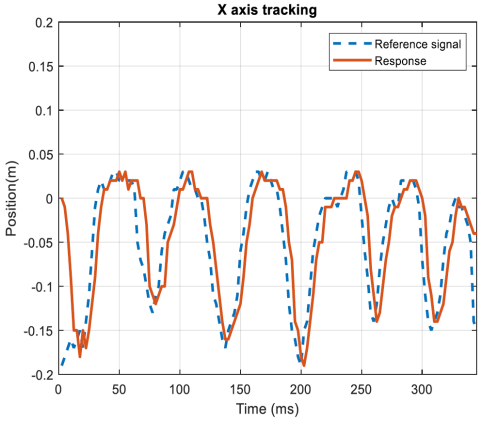


Figure 21 Object motion tracking on the y-axis (see online version for colours)

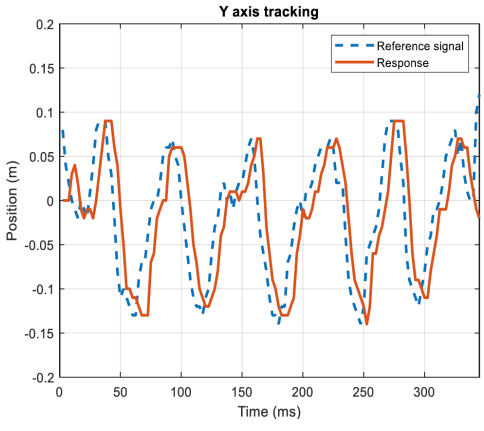


Figure 22 Detection of the human body (see online version for colours)

Figure 23 Upward hand gesture (see online version for colours)

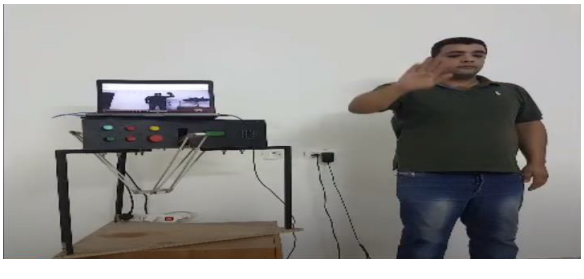


Figure 24 Left-hand gesture (see online version for colours)

Figure 25 Right-hand gesture (see online version for colours)

And the results for continuous random motion on the XZ plane are shown in Figures 26 and 27 respectively.

Figure 26 Hand gesture tracking on the x-axis (see online version for colours)

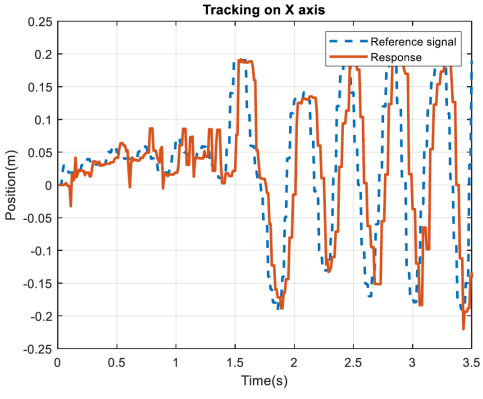
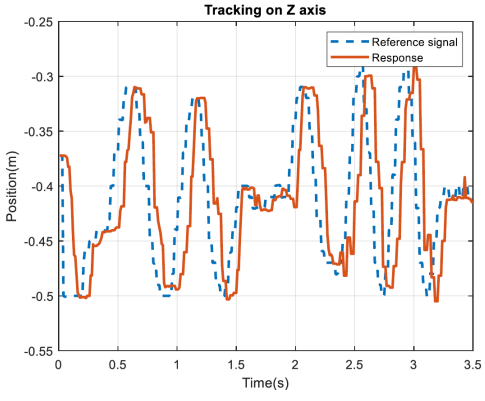


Figure 27 Hand gesture tracking on the z-axis (see online version for colours)



The results shown in Figures 20, 21, 26, and 27 ensure that the robot can track high-speed reference signals efficiently and in a real-time manner. It can be noticed that the response of the robot is smooth, fast, and has about no overshoot.

10 Conclusions

In this paper, two algorithms for real-time visual servoing had been implemented. The first one is used for tracking objects based on their colours and shapes which was implemented on a smartphone to avoid additional computational load on the main controller. This algorithm provides a solution for the field of view problem. The camera frame was located at the centre of the robot frame, while the offset between these frames was compensated using the LSA to generate transformation matrixes. More, this method does not require calibration.

The other algorithm was designed for hand gesture tracking using Kinect. This method provides the advantage of automatic calibration and generating transformation matrixes whenever the user requests.

Both methods were implemented experimentally and tested on the hardware prototype of the delta robot shown in Figure 1(b).

References

Agudo, A., Moreno-Noguer, F., Calvo, B. and Montiel, J. (2016) 'Real-time 3D reconstruction of non-rigid shapes with a single moving camera', *Computer Vision Image Understanding*, Vol. 153, No. 1, pp.37–54.

Andhare, P. and Rawat, S. (2016) 'Pick and place industrial robot controller with computer vision', *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, Pune, India, pp.1–4.

Asadi, K., Jain, R., Qin, Z., Sun, M., Noghabaei, M., Cole, J., Han, K. and Lobaton, E. (2019) 'Vision-based obstacle removal system for autonomous ground vehicles using a robotic arm', *ASCE International Conference on Computing in Civil Engineering*, Atlanta, Georgia.

Becker, B.C., MacLachlan, R.A., Lobes, L.A., Hager, G.D. and Riviere, C.N. (2013) 'Vision-based control of a handheld surgical micromanipulator with virtual fixtures', *IEEE Transactions on Robotics*, Vol. 29, No. 3, pp.674–683.

Bradski, G. and Kaehler, A. (2008) *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly Media, USA.

Brinker, J., Funk, N., Ingenlath, P., Takeda, Y. and Corves, B. (2017) 'Comparative study of serial-parallel delta robots with full orientation capabilities', *IEEE Robotics Automation Letters*, Vol. 2, No. 2, pp.920–926.

Bulej, V., Stanček, J. and Kuric, I. (2018) 'Vision guided parallel robot and its application for automated assembly task', *Advances in Science Technology Research Journal*, Vol. 12, No. 2, pp.150–157.

Chang, W-C., Cheng, M-Y. and Tsai, H-J. (2017) 'Image feature command generation of contour following tasks for SCARA robots employing image-based visual servoing – a PH-spline approach', *Robotics Computer-Integrated Manufacturing*, Vol. 44, No. 1, pp.57–66.

Correa, J.E., Toombs, J., Toombs, N. and Ferreira, P.M. (2016) 'Laminated micro-machine: design and fabrication of a flexure-based delta robot', *Journal of Manufacturing Processes*, Vol. 24, No. 2, pp.370–375.

Fang, X.L.a.X.Z.Y. (2011) 'Adaptive active visual servoing of nonholonomic mobile robots', *IEEE Transactions on Industrial Electronics*, Vol. 59, No. 1, pp.486–497.

Ficocelli, M. and Janabi-Sharifi, F. (2001) 'Adaptive filtering for pose estimation in visual servoing', *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Maui, HI, USA.

Gans, N.R., Dixon, W.E., Lind, R. and Kurdila, A. (2009) 'A hardware in the loop simulation platform for vision-based control of unmanned air vehicles', *Mechatronics*, Vol. 19, No. 7, pp.1043–1056.

Gridseth, M., Ramirez, O., Quintero, C.P. and Jagersand, M. (2016) 'Vita: visual task specification interface for manipulation with uncalibrated visual servoing', *2016 IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, pp.3434–3440.

Haddad, J.N., Rached, Z.S., Jajou, A.F. and Hage, R-M. (2019) 'On multiple regression diagnostics', *Applied Mathematical Sciences*, Vol. 13, No. 9, pp.415–421.

Hafez, H.M., Marey, M.A., Tolbah, F.A. and Abdelhameed, M.M. (2017) 'Survey of visual servoing control schemes for mobile robot navigation', *Scientific Journal of October 6 University*, Vol. 3, No. 1, pp.41–50.

Hu, C-Y., Chen, C-R., Tseng, C-H., Yudha, A.P. and Kuo, C-H. (2016) 'Visual servoing spanner picking and placement with a SCARA manipulator', *2016 IEEE International Conference on Industrial Technology (ICIT)*, Taipei, Taiwan, pp.1632–1637.

Li, C., Cao, C-q. and Gao, Y-f. (2017) 'Visual servoing based object pick and place manipulation system', *Current Trends in Computer Science and Mechanical Automation*, Sciendo Migration, Poland.

Li, C-L., Cheng, M-Y. and Chang, W-C. (2018) 'Dynamic performance improvement of direct image-based visual servoing in contour following', *International Journal of Advanced Robotic Systems*, Vol. 15, No. 1, pp.1–12.

Li, Y., Shang, D. and Liu, Y. (2019) 'Kinematic modeling and error analysis of delta robot considering parallelism error', *International Journal of Advanced Robotic Systems*, Vol. 16, No. 5, pp.1–9.

López-Nicolás, G., Özgür, E. and Mezouar, Y. (2019) 'Parking objects by pushing using uncalibrated visual servoing', *Autonomous Robots*, Vol. 43, No. 5, pp.1063–1078.

McClintock, H., Temel, F.Z., Doshi, N., Koh, J.S. and Wood, R.J. (2018) 'The MilliDelta: a high-bandwidth, high-precision, millimeter-scale delta robot', *Science Robotics*, Vol. 3, No. 14, pp.1–10.

Mills, S.J., Ford, J.J. and Mejías, L. (2011) 'Vision based control for fixed wing UAVs inspecting locally linear infrastructure using skid-to-turn maneuvers', *Journal of Intelligent Robotic Systems*, Vol. 61, Nos. 1–4, pp.29–42.

Mitsantisuk, C., Stapornchaisit, S., Niramitvasu, N. and Ohishi, K. (2015) 'Force sensorless control with 3D workspace analysis for haptic devices based on delta robot', *IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society*, Yokohama, Japan, pp.1747–1752.

Rehman, S.U., Raza, M.H. and Khan, A.R. (2019) 'Delta 3D printer: metal printing', *Journal of Electrical Engineering, Electronics, Control Computer Science*, Vol. 5, No. 3, pp.19–24.

Rouhollahi, A., Azmoun, M. and Masouleh, M.T. (2018) 'Experimental study on the visual servoing of a 4-DOF parallel robot for pick-and-place purpose', *2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*, Kerman, Iran, pp.27–30.

Sahu, U.K., Mishra, A., Sahu, B., Pradhan, P.P., Patra, D. and Subudhi, B. (2019) 'Vision-based tip position control of a 2-DOF single-link robot manipulator', *Proceedings of International Conference on Sustainable Computing in Science, Technology and Management*, Jaipur, India.

Sharida, A. and Hashlamon, I. (2019) 'Real time distributed controller for delta robots', *International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES)*, Karabuk University, Alanya, Turkey.

Sharida, A. and Hashlamon, I. (2020) 'Real time distributed controller for delta robots', *WSEAS Transactions on Systems and Control*, Vol. 16, No. 12, pp.99–107.

Shi, H., Chen, J., Pan, W., Hwang, K-S. and Cho, Y-Y. (2018) 'Collision avoidance for redundant robots in position-based visual servoing', *IEEE Systems Journal*, Vol. 13, No. 3, pp.3479–3489.

Singh, B., Sellappan, N. and Kumaradhas, P. (2013) 'Evolution of industrial robots and their applications', *International Journal of Emerging Technology Advanced Engineering*, Vol. 3, No. 5, pp.763–768.

Supriyanti, R., Setiawan, B., Widodo, H.B. and Murdyantoro, E. (2012) 'Detecting pupil and iris under uncontrolled illumination using fixed-Hough circle transform', *International Journal of Signal Processing, Image Processing Pattern Recognition*, Vol. 5, No. 4, pp.175–188.

Wachs, J.P., Kölsch, M., Stern, H. and Edan, Y. (2011) 'Vision-based hand-gesture applications', *Communications of the ACM*, Vol. 54, No. 2, pp.60–71.