Control of the Tiago robot using Pose Estimation and Position-based visual servoing

Stevedan Ogochukwu Omodolor[1] and Alberto Sanfeliu Cortes[1]

[1*]Mobile robotics and intelligent system department, Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Parc Tecnològic de Barcelona. C/ Llorens i Artigas 4-6,, Barcelona, 08028, Catalonia, Spain.

Contributing authors:
stevedan.ogochu.omodolor@estudiantat.upc.edu;
alberto.sanfeliu@upc.edu;

Abstract

In this paper, a strategy to control the position of the end-effector and the mobile base of the Tiago robot using visual feedback from a stereo camera is explained. To accomplish this, position based visual servoing technique is employed base on the estimation of position and depth information of the object been tracked. In order to ensure smooth position estimation, multple tecniques like median filtering and kalman filter was used.

Keywords: visual servoing, position based image servoing,kalman filter, median filtering

Introduction

The idea of visual servoing consist of controlling the motion of a robot using information obtained from a computer vision feedback system. The visual data can come from a camera placed directly on the the robot's end-effector(eye-in-hand configuration) or placed in a fixed position in the work space(fixed-camera configuration). The basic ideal of visual servoing consists of the minimization of an error value. The different visual-servoing techniques

differ with respect to the error function defined. In position-based image visual seroving(PBVS), the error function is specified based on position value in the robot task space. In Image-Based Visual servoing, the error function is defined in terms of image features [1]. Although, IBVS is considered favorable compared to PBVS due to it low sensitivity to camera calibration errors[2], PBVS method was used. This is due to the limitation with respect to the control layer of the Tiago robot.

1 Methods

The goal of this paper is implement a visual servoing strategy to track an object using visual feedback executed on the Tiago robot. To accomplish this, multiple control strategy is employed all of which is based on the 3D estimation of the object being tracked(Tube marked with Red color). With respect to the controller, two different control strategy are used: whole body controller to control the position of the whole robot body and a velocity controller for the based. It is worth mentioning that the implementation is implemented using ROS. In the following section, each of the element used during the implementation of this project is explained. A full schematic of the overall implementation can be seen in Figure 1. It consist of two different nodes: one responsible for the generation of the 3D pose of the object with respect to the camera link(tube_tracking_alg_node) and the other responsible for the control of the robot(visual_servoing_control_alg).

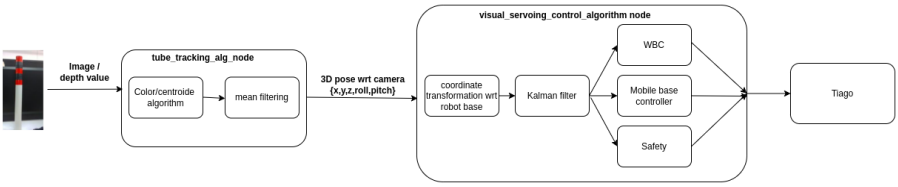


Fig. 1: Schematic of the code design.

1.1 Setup

During the experiment, the eye-in-hand configuration was used as shown in Figure 2. Appropriate cable length and size should be used during testing because it affects the resolution and range of view of the camera, which in turn affects the camera intrinsic parameter. This value is used during the computation of the 3D pose of the object being tracked. Although the current implementation takes this into consideration and further modification is needed in the code when a different cable is used.

Fig. 2: Camera location based on the eye-in-hand configuration

1.2 Tube tracking algorithm node

The goal of the section is to detail the necessary step in order to obtain the position in 3D space of the tube(Figure 3) using the realsense D435i camera. The is painted with color red to simplify detection.

Fig. 3: Tube detected

1.2.1 Pose estimation

To obtain the 3D pose, first color detection is used to isolate the red color from the background . To do this, the following steps were made: preprocessing, color detection, morphological operation and feature detection. An in-depth

explanation of each of these steps can be found in [3]. The resulting feature can be seen in Figure 4. This corresponds to the centroid and width of the three sections of the tube painted red. The point of interest been tracked is the centroid of the middle triangle.

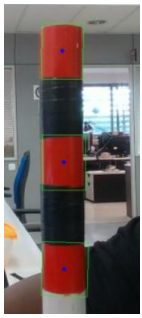


Fig. 4: Image after obtaining the feature point and contours

Using the technique explained [3], the depth value of each of the centroid is extracted for depth value outside of the working range of the realsense camera. With the depth value and the image pixel location of the two outer centroid, the roll and pitch of the tube with respect to the camera base link was estimated. All point are with respect the camera base link of the Realsense camera. With the depth and camera intrisics parameters(K), the point(u,v) of the image plane is projected to 3D space. This is done using the following equations:

$$x = \frac{(u - c_x)}{f_x}$$
$$y = \frac{(v - c_y)}{f_y}$$
$$X = depth.at(u, v)$$
$$Y = x * depth.at(u, v)$$
$$Z = y * depth.at(u, v)$$

where

- **u,v** position of point of interest in pixel
- c_x, c_y is the optical center of the camera in pixels(intrinsic parameter)
- f_x, f_y are the focal length of the camera(intrinsic parameter)

1.2.2 Mean filtering

To remove noise produced from the 3D pose estimation, a mean filter was used. It consists of replacing the 3D point obtained by the average of **n** previous 3d poses. A circular buffer(Figure 5) was used to implement the mean filter.

Fig. 5: Circular buffer

1.3 Visual servoing algorithm node

The visual servoing algorithm node is responsible for coordinating and managing the control system in order to generate the necessary command to send to the robot. To do so a state machine is used which contains the following states:

- **IDLE**: In this state the robot arm goes to the initial state(Figure 6), when the go_init_position is activated in the dynamic reconfigure GUI.
- **START_VISUAL_SERVOING**: In this state, the visual servoing algorithm is executed. It is activated only when, the initial position is reached and the start_visual_servoing box is checked in the dynamic reconfigure GUI
- **PRE_GRASP,GRASP, POST_GRASP**: When the perform_grasp box is activated in the dynamic reconfigure GUI, this states is activated in same order as stated. It moves the robot to grasp the tube
- **END**: When in this state, the robot returns back to ideal state.

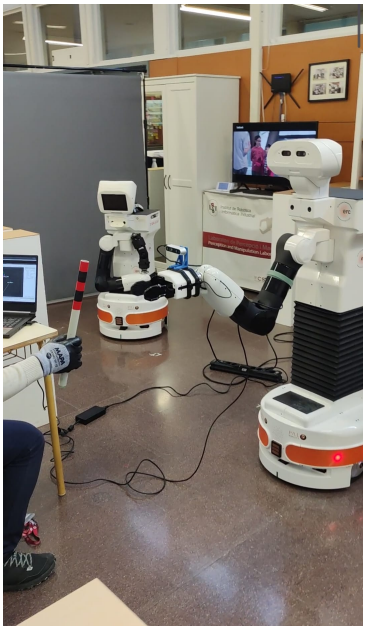


Fig. 6: Initial configuration of the robo

1.3.1 Visual servoing state

The 3d pose obtained from the tube tracking node is transformed from the camera base link to the base footprint node. This is because the whole body controller receives commands with respect to the base_footprint_link. In order to so, a static transform is added from the camera_link to the arm_tool_link. This is done in the launch file **"visual_servoing_alg.launch"**.

Kalman Filter

Kalman Filter is used to estimate the 3D pose of the tube when it cannot be measured directly due to sudden disappearance. The state vector **X** consists of the 3D position and velocity of the tube with respect to the robot base_footprint_link.

$$X = [x, y, z, roll, pitch, v_x, v_y, v_z, w_{roll}, w_{pitch}]$$

The measurement vector **Z** consists of the current position of the tube with respect to the robot base_footprint_link.

$$Z = [x, y, z, roll, pitch]$$

To build the state transition matrix, a strong assumption has been made that the tube move in a linear way with constant velocity.

$$\begin{pmatrix} I & T \\ 0 & I \end{pmatrix}$$

where

- **dt** is inverse of the frequency rate of the main control loop.
- **T** is the following matrix:

$$\begin{pmatrix} dt & 0 & 0 & 0 & 0 \\ 0 & dt & 0 & 0 & 0 \\ 0 & 0 & dt & 0 & 0 \\ 0 & 0 & 0 & dt & 0 \\ 0 & 0 & 0 & 0 & dt \end{pmatrix}$$

- **I** being a 5 x 5 matrix

The observation model matrix **H** is :

$$\begin{pmatrix} I & 0 \end{pmatrix}$$

The process noise covariance matrix **Q** is

$$\begin{pmatrix} E_x & 0 & 0 & 0 & 0 \\ 0 & E_y & 0 & 0 & 0 \\ 0 & 0 & E_z & 0 & 0 \\ 0 & 0 & 0 & E_{roll} & 0 \\ 0 & 0 & 0 & 0 & E_{pitch} \end{pmatrix}$$

The error values are found experimentally. The kalman filter is implemented as follows, when the **tube** is in frame and has not being previously detected for a fixed amount of time(fixed in the code), the kalman filter is reinitialized. On the contrary, if object is in frame and has been previously detected for a fixed amount of time, the kalman filter is updated. The prediction process is done regardless of the object been in frame or not but under the condition that it has been detected for the fixed amount of time mentioned previously or has been reinitialized. A pseudo-code can be seen in alg. 1.

Algorithm 1 Kalman filter implementation

```
found ← false
while visual servoing activated do
    if found then
        perform prediction
    end if
    if object in frame then
        ...
        if not found then
            reinitialize timeout
            reinitialize state
            found ← true
        else
            update kalman filter
        end if
    else
        Update timeout
        if timeout reached then
            found ← false
        end if
    end if
    if found then
        control_strategy()
    end if
end while
```

Control strategy

In order to control the robot, different strategy are used as shown in Figure 1. First, in the safemode, the robot functionalities are shut down completely, both base and body. This is activated when there is a large change in position value of the tube. The other two controller, mobile base speed controller and the WBC, are used based on the position of the end-effector. A a 3D virtual box constraint in placed with respect to the initial position of the robot shown in Figure 6.

The robot uses the WBC to control the arm and the torso, when the arm is within the 3D box constraint. The WBC is a quadratic hierarchical solver that provided the inverse kinematics of the whole robot body. It performs different tasks based on priorities. In the case of the Tiago robot, the tasks with the highest priority are joint limit avoidance and self-collision avoidance. This means that the user can command the robot's end-effector to go to a 3d point in space but this task has lower priority.A module "**tiago_wbc_controller_modules/wbc_body_module**" has been created to make it easier to interact with the WBC. It allows the user to move the robot in a straight line or perform a ptp trajectory. One issue with the WBC is that

the response of each axes is different, this is probably due to the task priority. For example, the x axes response is slower compared to other axes. In order to combat this problem, the WBC module is implemented as follows: when the end-effector is close to the goal(error), the robot arm stops. Due to the different response in the axes, the error can be specified for each axes.

```
bool moveTo(geometry_msgs::PoseStamped goal, bool
    straight_line = false, bool check_x = true, bool
    check_y = true, bool check_z = true, bool
    check_orientation = false);
```

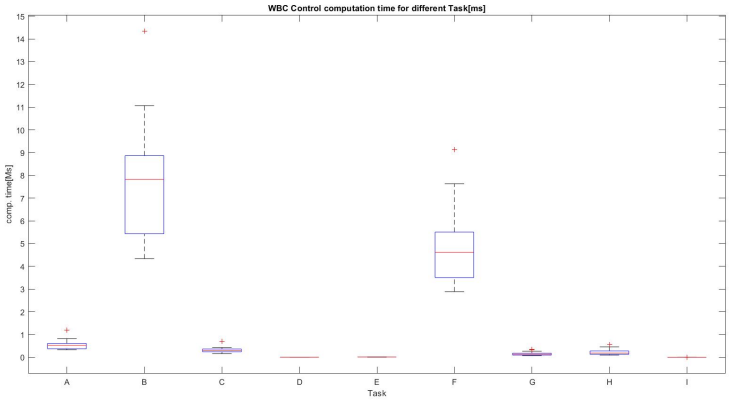
When the robot end-effector is out of the range of the 3D virtual box, the base is controlled. It consists of a velocity controller where the commands are rotaional and translational speed. A video demostration can be seen in link to demo.

2 Computation time Study

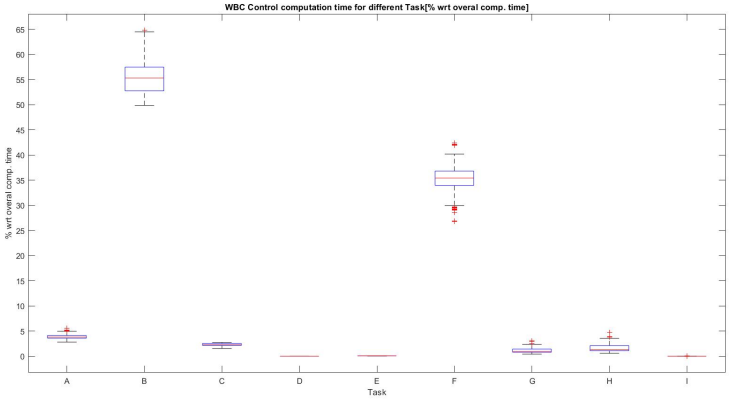
To improve the system response time, a study of the computational time of different task during the code was recorded. It is worth noting that the computational time for the display of the tube tracking node was added. Although, this result is not taking into consideration during the computational study because in a typical working demonstration, the display functionality would be deactivated. It is used in this case for debug purposes.

The list of task of which the computational time was measured are as follows:

- **A**: Time convert Ros image to opencv math
- **B**: Time find red color and obtain denoise image
- **C**: Time compute contour and 3d from 2d
- **D**: Time buffer computation
- **E**: Time between buffer computation and end of vision node
- **F**: Time convert Ros image to opencv math
- **G**: Time convert Ros image to opencv math
- **H**: Time convert Ros image to opencv math
- **I**: Time convert Ros image to opencv math

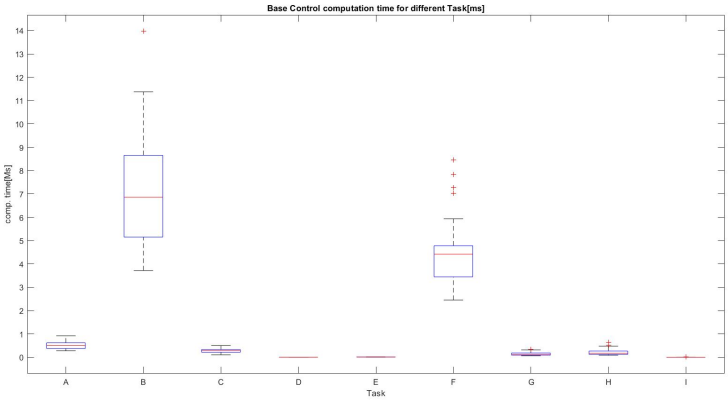


(a) WBC controller computation time[MS]

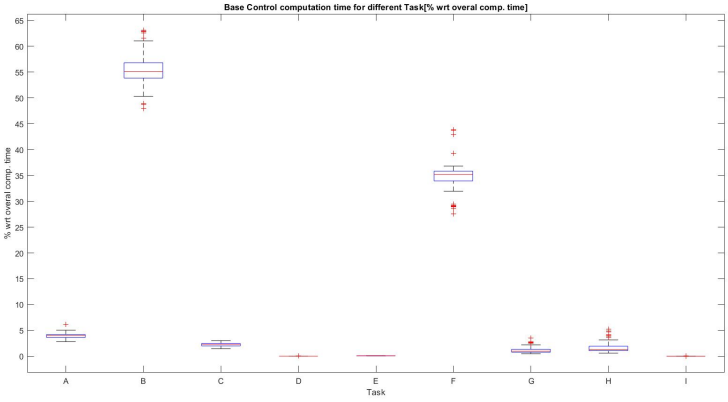


(b) WBC controller computation time[%]

Fig. 7: WBC controller computation time

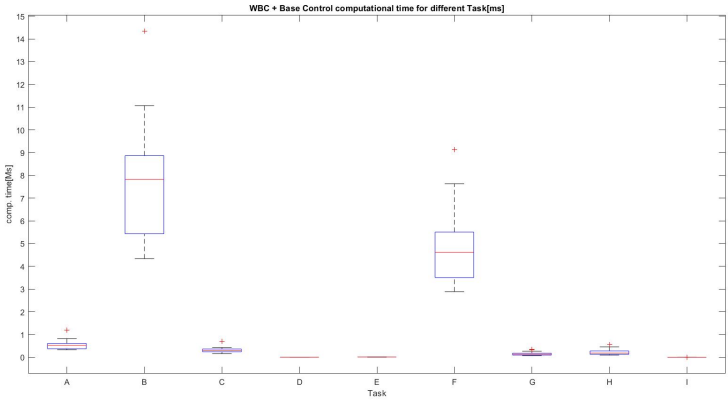


(a) Base controller computation time[MS]

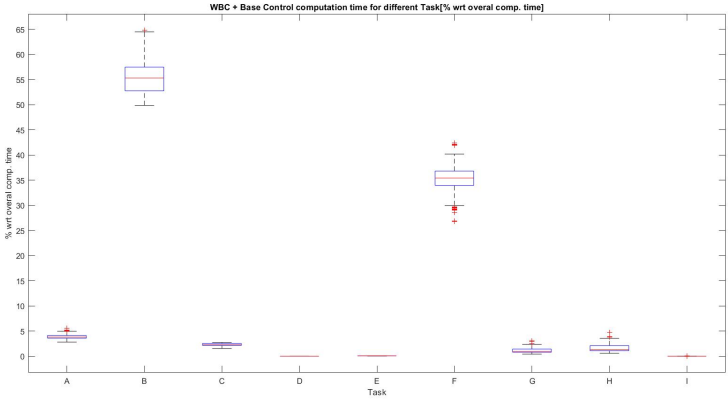


(b) Base controller computation time[%]

Fig. 8: Base controller computation time



(a) WBC + Base controller computation time[MS]



(b) WBC + Base controller computation time[%]

Fig. 9: WBC + Base controller computation time

As we can be observed, the task that takes longer is the task B, which in almost all the different controls, takes 60 % of the overall computational time. This makes sense because during this task, a lot of matrix operation is done on the image. The other task(excluding the display task) have similar computational time. As the graph shows, the computational time is not influenced by the type of controller used(WBC/base). Another possible reason which could cause the system to have a slow response is the reaction of the controller. Although this has not being tested yet. The data results and more graph can be found in the file **visual_servoing_data_computation_time**

3 Conclusions

In this report, the algorithm needed to implement a virtual servoing using the tiago was explained. It included both the detection and the robot control. Although the system implemented works, improvement is needed to increase the system's reaction time.

References

[1] Cheng, E.D.-L.G.: A new method for solving 6d image-based visual servoing with virtual composite camera model. IEEE-RAS International Conference on Humanoid Robots (2014)

[2] Michels, T., Hochländer, A.: "a tutorial on visual servo control. TU Darmstad (5), 651–670 (1996)

[3] Omodolor, S.O.: Position and depth estimation of the tube for visual-servoing (2021)