Object Tracking Robot using Stereo Vision

Anonymous ACVRW submission

Paper ID 20

Abstract. *This paper originates from a coursework done in the module Applied Robotics at the Vorarlberg University of Applied Science (FH Vorarlberg). The work is about finding a green laser point on a white flip-board chart. A robot with a 3D camera system tracks this point and drives its laser-pointer tool to the same spot. The premiss was not to use any position controller and instead compute the desired axis-angles from the 3D information of the found laser dot. The paper describes the design and the implementation of the system.*

1. Task Constrains

The following requirements were posed to fulfil this coursework:

- Create a 3-axis robot and not a simple two-axis pan/tilt device.
- Measure the 3D point of the laser and drive to its position without using a closed loop controller.

2. Design

The robot consists of 3 Dynamixel MX-64AT motors, two Microsoft Lifecam HD 3000 (640x480 pixel) and a laser-pointer. Aluminium frames are attached to the motors to build the framework of the robot. The cameras and laser-pointer are connected with aluminium brackets. The design of the robot and its components is established with SolidWorks. Figure 1 shows a 3D render of the fully assembled robot.

3. Robot Kinematic

The motors can be controlled by a supplied library, which is written in C [6]. This allows us to write our own DLL to implement the backwards kinematics and control the robot as a whole. Figure 2 shows

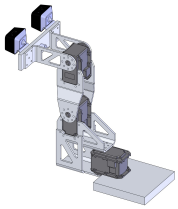

Figure 1. 3D render of the assembled robot

the coordinate systems that are used to get the link parameters shown in table 1. The link parameters are established using the Denavit Hartenberg convention mentioned in [1, p. 70-79].

With the link parameters, the inverse kinematics matrix can be generated. This is used to transfer the 3D point captured by the camera, from the tool frame $\{F\}$ to the base coordinate system $\{0\}$. From there, the point can be transferred into the coordinate frame $\{3\}$. This transformation is implicitly included in equation (2). Together with (1) these equations can be used to calculate the desired robot configuration. Because θ_1 and θ_2 directly depend on each other, θ_1 can be set to any arbitrary angle.

$$\theta_0 = arctan\left(\frac{P_{Y0}}{P_{X0}}\right) \tag{1}$$

$$\theta_2 = -arctan\left(\frac{P_{Z0} - 100 - 100cos(\theta_1)}{\sqrt{P_{X0}^2 + P_{Y0}^2} - 50 - 100sin(\theta_1)}\right) \tag{2}$$

i	α_{i-1}	a_{i-1}	d_i	θ_{i-1}
1	0	0	100	θ_0
2	-90	50	0	$\theta_1 - 90$
3	0	100	0	$\theta_2 + 180$
F	90	0	68	0

Table 1. link parameter

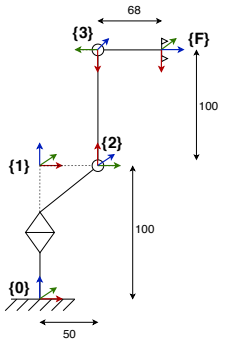

Figure 2. axis coordinate systems for computing the inverse kinematic

4. Tool Calibration

The tool calibration includes three steps: Estimating the camera projection matrix as well as the transformation from the main camera (left in robot sight direction) to the TCP and the transformation of the laser pointer towards the TCP.

4.1. Hand-Eye Transformation Matrix

The hand-eye calibration is measured from the real robot. It is assumed, that the rotation between the camera coordinate system and the tool is small. We only have to rotate the coordinate system and take the translation vector into account. Another method would be that of Park and Martin [3] which is also provided in OpenCV.

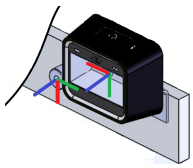

Figure 3. axis coordinate systems for computing the backwards kinematic

4.2. Laser Calibration

Since the laser is designed to lay directly in line with the Z-coordinate of the tool frame, there is no calibration needed. This demands a fine level of accuracy for the manufacturing process.

4.3. Stereo Camera Projection Matrix

The camera projection matrix is estimated from multiple images of the same chessboard calibration plate [5, p. 178-182]. At first, the intrinsic parameters of both cameras are estimated separately using the OpenCV function *calibrateCamera*. With this information the function *stereoCalibrate* can be used to estimate the translation and rotation between the two cameras. With these matrices and the function *triangulate* a 2D point correspondence can be associated with one 3D point. The camera calibration app is implemented in Python using these OpenCV functions [2].

5. Object Detection

Two methods for detecting the laser dot are tested. **Option 1:** use a HSV-threshold and only look for pixels with that specific value. This method is unstable against different illumination scenarios.
Option 2: use template matching with user-defined template. The main idea is to calculate a normalized difference between every pixel in the image and a template [4, p. 1061-1062]. This function is called *matchTemplate* and is also provided by OpenCV. The method *CV_TM_CCOEFF_NORMED* was found to work most reliable for multiple different lighting scenarios. This method uses the equation 3 below, where T stands for the template and I is the image.

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}} \quad (3)$$

6. Result and Conclusion

We can show that the described system works as specified. We find, that template matching is better suited to detect small objects in a constantly changing image rather than searching the images for a specific colour. Further steps could include to verify the assumption about the orientation of the laser tool as well as refining the hand-eye calibration.

References

[1] J. Craig. Introduction to robotics mechanics and control, 2018.
[2] OpenCV. *The OpenCV Reference Manual*, 4.1.1 edition, July 2019.
[3] F. C. Park and B. J. Martin. Robot sensor calibration: solving AX=XB on the Euclidean group. *IEEE Transactions on Robotics and Automation*, 10(5):717–721, Oct 1994.
[4] R. E. W. Rafael C. Gonzales. Digital image processing, 2018.
[5] A. Z. Richard Hartley. Multiple view geometry in computer vison, 2008.
[6] ROBOTIS-GIT. Dynamixelsdk. `github.com/ROBOTIS-GIT/DynamixelSDK/tree/master/c`, 2019.