Target Localization through Image Based Visual Servoing

Peter A. Greczner
Cornell University
Masters of Engineering 2010
pag42@cornell.edu

Matthew S. Rosoff
Cornell University
Masters of Engineering 2010
msr53@cornell.edu

Abstract – This projects implements Image Based Visual Servoing techniques to localize an AX-12 arm gripper in relation to a target object. The AX-12 is a multi-joint robotic arm that can be manipulated to perform a variety of tasks such as grasping. In such a task as grasping the target in question is usually not known apriori and computer vision or other methods are needed to locate such an object. Image based visual servoing is a method that uses computer vision algorithms to detect a target object and manipulator object in the image frame. Based on an error function relating the estimated poses of manipulator and target object it attempts to move the manipulator so that the error function minimizes. This project defines the error function so that it minimizes the image frame x and y Euclidian distance between manipulator and a sphere points. The manipulator is located using SURF and the target sphere is located using a combination of thresholding, edge detection, and a circular hough transform. Implementation of the algorithm is done in ROS as a set of three services, one that publishes the location of the target, one that publishes the location of the manipulator, and a final service that communicates with the AX-12 to move it to minimize the error function. Our project successfully manipulated the robotic arm to the target object with sub-inch accuracy.

I. INTRODUCTION

Visual servoing is a field of machine vision and control that seeks to use video feedback to control motion of a robotic system. Visual servoing can be decomposed into 3 problems – Find objects in the camera reference frame, convert coordinates to the global frame (if necessary), and plan a path for movement in the global frame. Visual Servoing is a feedback technique, so it can be robust to some uncertainties in formulating coordinates, scaling, and dynamics in the problem. Additionally, visual servoing does not require very specialized sensors or instrumentation – just a video camera and a processing unit. Visual servoing must make up for sensing inaccuracies using software. Consequentially, visual servoing algorithms are plagued by many of the same problems most computer vision algorithms face – noise, distortions, lighting and environmental variability and computational intensity. Visual servoing is a new technology and was only first proposed in 1996 by S. A. Hutchinson, G. D. Hager, and P. I. Corke (1) and significantly improved upon in 2006 and 2007 by F. Chaumette and S. Hutchinson (2). In 2006, visual servoing was formalized into 2 distinct approaches – Image Based Visual Servoing (IBVS) and Position Based Visual Servoing (PBVS). Image based visual servoing seeks to minimize errors in the image plane of the camera while position based visual servoing seeks to calculate then minimize errors in the global reference frame.

II. CURRENT PROBLEM

Many robotic applications require some form of manipulation of the robot based on environmental data. One of these applications is robotic arm manipulation in which the goal is to grasp or follow a target object. While inverse kinematics can guide a robotic arm to a desired position of an object, a problem arises when that target object moves or its position is unknown ahead of time.

In these situations it is necessary to use other sensing techniques, such as laser data, vision algorithms, and possibly ultra-sonic information. These techniques create a more detailed picture of the unknown environment.

III. PROPOSED SOLUTION

This project solves the aforementioned problem via an implementation of Image Based Visual Servoing (IBVS). IBVS attempts to solve this problem by using monocular image data to locate a target in an image and guide the manipulator towards the target.

In our design, we wrote software layers to allow a claw to grab an object using just vision feedback. Our design focuses heavily on modularity such that any future contributors can improve, update, or adapt the project to suit their individual needs. Modularity is achieved through ROS nodes and adaptable parameters. IBVS is inherently robust to changes in the environment, so long as the coordinates it receives remain accurate. There are 4 layers to our software – image capture, object and claw detection, high-level motion control, and low-level motor control – all running in a message-passing ROS package. The arm is already built by Crust Robotics and the low-level motor controller is already written by the University of Arizona in ROS (smart_arm_controller package). Image capture is performed using openCV with a video feed coming from GStreamer in Linux. Object and claw detection nodes receive the images through a topic subscription in ROS and produce image plane Cartesian coordinates. These coordinates are passed through a topic to a high-level motion controller (IBVS) that will motion plan and feed angular coordinates to the low-level *smart_arm_controller* for motor actuation. The change in environment will be detected by the video camera, closing the feedback loop. Any of these nodes can be modified to implement different algorithms, providing modularity and flexibility in implementation and testing.

A. Environment Details

This project was performed in the Robot Learning Lab of Cornell University. The robotic arm that was used was an AX-12 Crust Crawler (fig 1.). It is a multi-joint robotic arm that uses servo motors as a means of to manipulate the arm. Communication with the arm is done via serial commands to specify an absolute servo position of each motor; however this communication layer is implemented by the University of Arizona in their *smart_arm_controller* ROS package. Writing these commands entails setting up a topic in ROS, such as *shoulder_pan_controller/command* and publishing to that topic.

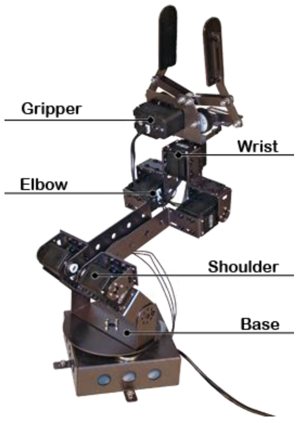

Fig 1. AX-12 Arm

The video was captured using a Microsoft webcam that had an image resolution of 1280x800 pixels. This image information was captured using the ROS *Probe* package by Brown University.

Image processing on the incoming video stream was done using OpenCV. The *Probe* package publishes the image data as an IplImage that is native to OpenCV and then OpenCV processing utilities can be used to extract appropriate information and manipulate the image.

The testing environment was constrained to have solid color background in the back of the scene. This was done in order to minimize the scene noise as much as possible so work could focus on algorithm implementation as opposed to dealing with a noisy environment.

B. Object Detection

Detecting objects in the image frame is a central component of Image Based Visual Servoing. There are two objects that have to be detected. The first is the manipulator object- in particular the grasping point of the manipulator. The second is the target object to move your manipulator towards.

Typically, the geometry of the target object is known apriori. The geometry and other information relating to the manipulator grasper is also known.

In this project the target to be detected in the image is a solid orange colored sphere. The manipulator is the AX-12 claw.

1) Detecting the Target Sphere: Detecting the target sphere was done by first color thresholding the image to leave only the regions with high probability that they were the sphere. This was done by manually gathering RGB color data on the sphere in each testing setting and threshold within one standard deviation of each RGB.

```
pixel_score = 0
pixel_score += (Pixel(I,J).R > mu.R – std.R
&& Pixel(I,J) < mu.R + std.R) ? 1 : 0
pixel_score += (Pixel(I,J).G > mu.G – std.G
&& Pixel(I,J) < mu.G + std.G) ? 1 : 0
pixel_score += (Pixel(I,J).B > mu.B – std.B
&& Pixel(I,J) < mu.B + std.B) ? 1 : 0
Pixel(I,J) = (pixel_score == 3) ? Pixel(I,J)
: 0
```

After this step the image is converted to grayscale color, canny-edge operated, and a Circular Hough Transform is performed. The radii that are checked are based on what the previous round radius was. This is similar to setting a region of interest in an image, only it sets radii of interests to speed up computation.

Based on the list of returned circles the circle that is determined to be the target sphere is the circle whose color around the center best matches the target color.

The target finding program then publishes the image frame's x-coordinate, y-coordinate, and the radius of the found sphere.

2) Detecting the Manipulator Object: Detection of the manipulator object is done by using SURF. SURF stands for Speeded Up Robust Features and is used to detect scale and transform invariant features in an image. More details on SURF are discussed in (6). We apply SURF to a basis image of the claw and develop a vector of the features. This vector can then be used to compare to all the features that are found in each frame of video that is processed. Nearest neighbor correlation is then performed between the basis feature and the current frame's feature to detect the manipulator location and pose. After features have been related between images the object is located.

OpenCV was used to perform the SURF feature extraction. It was speeded up by developing a region of interest (ROI) around the probably manipulator. When the algorithm is confident it has found the grasper it sets the ROI within 200 pixels of the center of the grasper, creating a 400x400 pixel image, which approximately 1/6th the size of the input image. If the algorithm loses the grasper at any point, the ROI is expanded back to the full image size until the grasper is found again.

The grasper finding program outputs the x-coordiante and y-coordinate in the image frame along with a scaling factor. The scaling factor is related to the dimensions of the grasper and is used to get an idea of the elevation of the grasper. This helps in making the grasper co-planar with the target sphere.

C. Image Based Visual Servoing Implementation

After the objects have been detected and relevant coordinate information in the image frame along with pose information (radius, scale factor, etc.) has been found, the image based visual servoing (IBVS) algorithm runs.

IBVS works by trying to move the manipulator so that a user-defined error function E(X) is minimized. Where X is state information of the system. In this project the state of the system are the (X_{sphere},Y_{sphere},R_{sphere}) and ($X_{grasper}$,$Y_{grasper}$,$Scale_{grasper}$). The user-defined error function is to minimize the (X_{sphere} – $X_{grasper}$) in the same plane of the environment.

The IBVS program then works by subscribing to the sphere topic and the grasper topic. It calculates E(X) and calculates a dTheta, which is an angular velocity for how the shoulder of the arm should move in order to minimize E(X). dTheta is proportional to the signed magnitude of E(X). Thus, for situations in which the grasper is far away from the sphere, the arm will move quicker, and as it approaches close to the sphere, the arm will then move very slowly.

D. ROS Architecture and Services

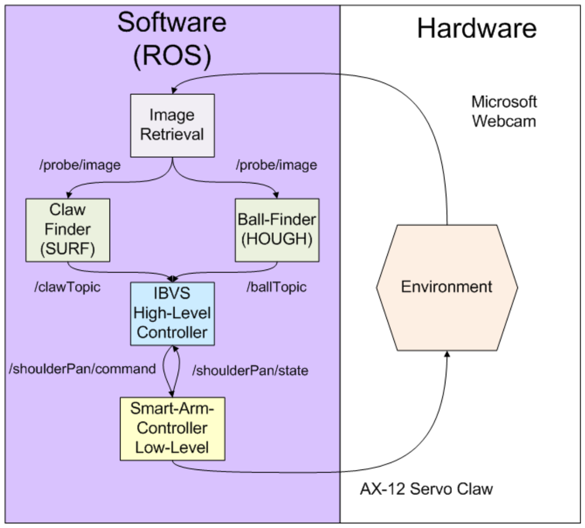

Fig. 2. This image shows the general outline of ROS services and functionality.

As shown in the figure above, all functionalities reside in nodes in ROS. ROS is a Robot Operating System that supports modular, message-passing code. Our nodes are the Image retrieval, Claw Finder, Ball-Finder, IBVS controller, and low-level smart arm controller. We wrote the IBVS, claw finder, and ball finder nodes. The other nodes were provided to us from the University of Arizona in a ROS package. This package is called “smart_arm_controller” under the “ua_controllers” designation in the “ua-ros-pkg stacks” ROS stack. This amazing stack can be retrieved directly from the ROS website or via SVN. The UA stack has so much support and information on the AX-12 arm –more so than the CrustCrawler website. ROS has a unique message-passing structure that supports HTTP message passing. This means that each node can run on any machine (except Image Retrieval and smart arm controller because they communicate directly with physical hardware). We experimented with turning the lab into a ROS cluster such that each computer can process one node. This lead to a substantial speed-up when properly launched; however there was some limitation when passing large messages with the image frames in them. Our code uses topics wherein nodes publish and subscribe (as shown in the chart above).

IV. RESULTS

The algorithms were able to work successfully and deduce where both grasper and target sphere were. The SURF algorithm to find the grasper worked approximately 70% of the time. The target sphere detector was heavily dependent on lighting and color information. However, this only required a onetime analysis of the current environment during each testing session to recalibrate the color information for it to work correctly.

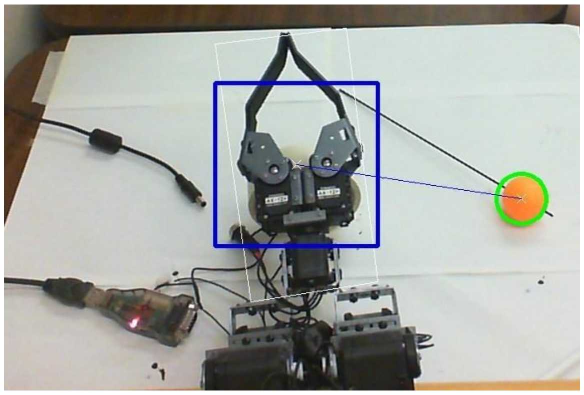

Image 1. This image shows what one frame of the IBVS algorithm detects.

Image 1 shows how the IBVS algorithm processes each frame of the image. It receives the information from the target and claw services and draws on the image a visual cue for what information is being sent to it. The blue square represents the best guess for the central region of the grasper, and the white parallelogram is a best fit of the basis image to the current frame. The green circle is an outline of the target sphere. A line is drawn from the center of the grasper to the target to show the current error between the two positions. The IBVS then takes this error, as described in section III.C and moves the grasper towards the ball.

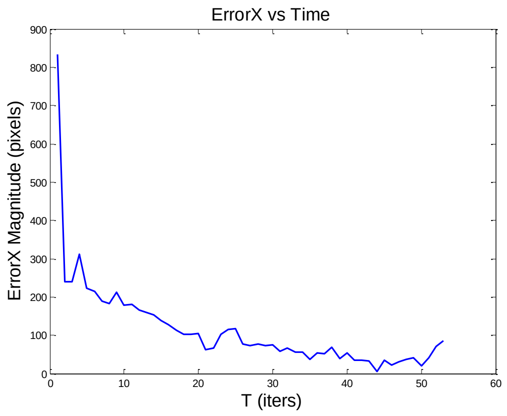

Fig. 3. Graph of E(X) over time, where time is the number of iterations that the manipulator was moved towards the ball.

Figure 3 is a plot of the X component of the Error vector against the iterations of our control loop (~25 sec). Notice that the error tends toward 0 in time, but hovers around 20 pixels in the image frame. This indicates an error of about 0.75” in the global reference frame. We anticipate the error to approach 0, however there are reasons why this may not happen exactly. Delays in the control loop cause small oscillations seen in the graph above. Additionally, we are using a simple 1st order proportional controller without integral control to drive the error slowly toward 0. We feel that the error shown above is acceptable for many applications (0.75” accuracy is decent in computer vision).

V. CONCLUSIONS

Our project worked incredibly well and exceeded our expectations and goals. The project minimized the error between the claw and the ball down to sub-inch accuracy. This is verified by looking at the error plot below and noticing that the error converges to ~20 pixels in the image frame. This error is approximately 0.75” in the global reference frame. Our project is also highly modular-not only can a user swap different algorithms in easily without affecting the entire project, but the project can also run across a cluster of computers for speedup.

If we had more time, we would have liked to implement one of the other arm teams’ inverse kinematics package so that we could minimize error in the image-y direction better. Also, we would have liked to take more physical error data to do a full statistical analysis on the accuracy and precision of our algorithms.

VI. OTHER NOTES

The code is in the Personal Robotics repository. It is located in the applications folder under demo_visual_servoing. There are three executables that need to be run.

The first is cameraclick which is used to locate the ball. It takes two parameters. The first is used for the edge detector and is an integer. 50 is a good choice. The second is the image topic, and that should be /probe/image.

The second is Claw, which is used to locate the claw. It also takes three parameters. The first is either a 0 if you want to display the results in an image window, and 2 if you do not want to open that image window. The second parameter again should be /probe/image.

The third executable is IBVS which manipulates the arm and runs the visual servoing algorithm. The other two executables should be started before this is run for best performance, but it isn’t completely necessary. This also takes three parameters. Same details as with Claw.

If you have issues downloading and running from the repository, please contact pag42@cornell.edu. I put it in the repository and am not sure if I did it right because it was called one package on my account, but that package already existed in the repository so I changed the name of the folder and added that. This is because we started from a base package called demo_ax12_camera which returned a camera feed written by Dan. We continually modified that for our project and since I didn't want to overwrite his updates to it in the repository, I went and changed our folder to demo_visual_servoing.

One last note for the professor – We extended the implementation of both the Claw detector and the ball detector to use intelligent ROI in order to speed up computation time, so where there was a lag in the demo video, now the Claw grasper is up to around 6-10 frames per second, which is much better than the 1-2 frames before. It seems a little bit more accurate now too since it is looking at a smaller region of data. Figured I'd add that in at the end since you seemed interested in the robustness of the detection at the poster presentation.

Overall, we enjoyed working on this project and would like to thank you for help when we had to switch to a completely new project topic right at the midterm deadline.

REFERENCES

[1] S. A. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. IEEE Trans. Robot. Automat., 12(5):651--670, Oct. 1996.

[2] F. Chaumette, S. Hutchinson. Visual Servo Control, Part I: Basic Approaches. IEEE Robotics and Automation Magazine, 13(4):82-90, December 2006

[3] P. Martinet, J. Gallice. Position based visual servoing using a non-linear approach. LASMEA, Universite Blaise Pascal, UMR 6602 du CNRS.

[4] K. Deguchi. Image Based Visual Servoing with Partitioned Approach. Graduate School of Information Physics, Toboku University.

[5] D. Kragic, H.I. Christensen. Survey on Visual Servoing for Manipulation. Centre for Autonomous Systems, Numerical Analysis and Computer Science, Fiskarorpsv. 15.A.

[6] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, Speeded-Up Robust Features (SURF), Computer Vision and Image Understanding, Volume 110, Issue 3, Similarity Matching in Computer Vision and Multimedia, June 2008, Pages 346-359