RoVi1 Final Project

Fall 2017

1 Background

Closed loop online control of a robot is often referred to as servoing. When the feedback in the control is based on visual information, we refer to it as visual servoing.

Visual servoing applications can be divided into image based and position based depending on whether the error function of the control is formulated in image space or in Cartesian space. Furthermore visual servoing can be divided into eye-in-hand and eye-to-hand depending on whether the camera is mounted on the robot (eye-in-hand) or next to it (eye-to-hand). Each of these four combinations has certain drawbacks and advantages and need to be chosen based on the application.

2 The Project

In this project you are going to implement an eye-in-hand image based visual servoing solution. The project contains three parts:

- **Feature Extraction:** Implementation of a feature extraction algorithm which can be used to find and track features in the image.
- **Tracking:** Implementation of the robot control based on the inverse Jacobian algorithm. Notice that the algorithm needs to be adapted to take inputs in image space rather than in Cartesian space.
- **Integration and Test:** Integration of feature extraction and tracking, and testing and verification of the algorithms in a simulated environment.

On BlackBoard under the RoVi1 course you will find a folder called Final Project containing:

- This project description.

- The real world vision marker sequences.
- The workcell (with a PA10 robot and the environment).
- A sample plugin for RobWorkStudio (including backgrounds, markers and motions).

The plugin shows how to set a texture on the marker in the scene, how to grab an image from the simulated camera and how to use a timer for setting up a control loop running with a fixed sample-rate. The sample plugin for RobWorkStudio includes both RobWork, RobWorkStudio and OpenCV in the CMakeLists.txt. You should just cmake and import the project into QtCreator as done in the robotics lectures.

Remember that this is a *sample* plugin, so you should correct it according to your needs in the project. In the SamplePlugin.cpp/.hpp files you will find most of the code needed including a timer() function which is where you will implement your vision and robotics code. The ui_SamplePlugin.h file maps between the plugin and the GUI. From the SamplePlugin.ui it is possible to design the GUI in QtCreator (clicking the file in QtCreator). The plugin has two buttons; the first button loads the marker and background (paths specified in the source file) and the second replaces Lena with the camera image (try to jog the robot or the marker around). Ignore the prompt about not finding the executable when compiling in QtCreator. A library is generated in the lib folder.

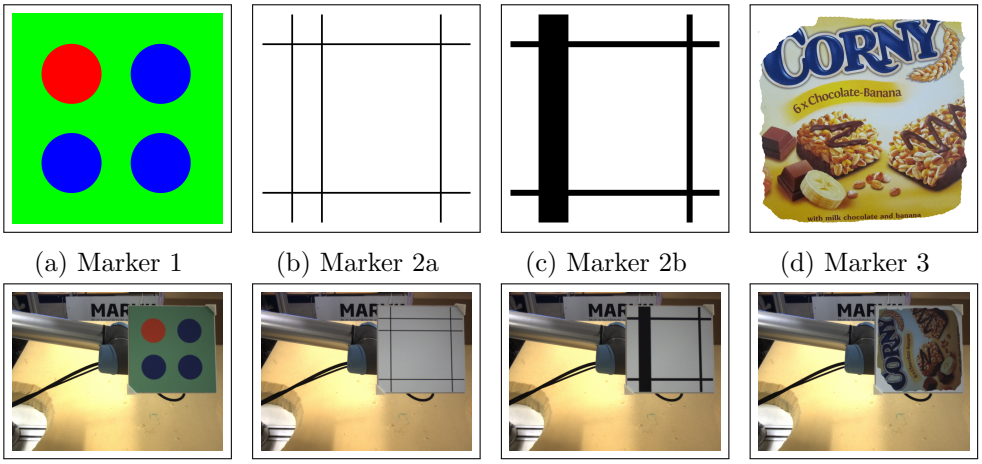
3 Feature Extraction

To minimize the difference between a target position and the current position we need to define an error function. This error is normally based on a small set of distinct points (e.g, 1–4) in the image. Instead of detecting these points directly it is usually better to have a more complex marker that allows for a more stable derivation of individual points.

To be able to form a closed loop visual serving system (see Section 5) in our case one needs to detect these markers (and thereby derived points) from simulated images. Unfortunately this hides most of the challenges on the vision side. Therefore we provide images sequences that allow you to deal with real images (see e.g., Figure 1). For each marker we provide an easy sequence and a hard sequence. Your evaluation should be done with the easy one as well as the hard one. It is not essential that your method works with every single frame, just document the performance.

For the provided sequences the following aspects could be good tools:

A) Color (e.g., color segmentation)



(a) Marker 1 (b) Marker 2a (c) Marker 2b (d) Marker 3

(e) Sequence 1 (f) Sequence 2a (g) Sequence 2b (h) Sequence 3

Figure 1: Depictions of markers and real sequences.

B) Simple features (e.g., derived from area or outline)

C) Edge detection

D) Point features (e.g., SIFT features together with homography estimation)

Example approaches could be:

1. Color segmentation, blob (connected component) detection, classification of blobs to detect circular/elliptic blobs, (one blob's color is different to allow for the assignment of corresponding blobs). Computing the blobs' centre of gravity and deriving marker points. This approach covers aspects A and B. This would work with the marker in Figure 1a.

2. Detection of lines in the image using the Canny edge detector and the Hough transform. (Rejection/selection of lines based on the color on each side). Combining lines to find possible intersections. Verifying the marker by looking at the distances between intersections. This approach covers aspect C (and B). This would work with the markers in Figures 1b and 1c.

3. Extracting SIFT features. Matching the extracted SIFT features against SIFT features in a marker reference model. Using RANSAC to find a homography that aligns the reference model with the observations. Use the homography to transform the marker reference points from the model to the image. This approach covers aspect D. This would work with the marker in Figure 1d.

While you are certainly free to select one or two of the above suggestions this project allows for the exploration of very different approaches and also additions (e.g., tracking, different combinations). Please make sure that you do not use mostly the same approach for all your markers (at least two different approaches (of the above or additional ones) should be used).

Your task is to implement a marker finding/tracking approach for at least two of the provided sequences/markers (2a and 2b are slight variations on the same problem, so selecting 2a and 2b does not constitute two makers). We expect you to evaluate the methods' performance in detail. Aspects that should be certainly analyzed are: runtime (how long does your method take to find the marker, what is the variation), precision (e.g., binary: is the marker found at the right position? or discrete: how big is the error in pixels? for each frame). Additional aspects here can be: Sensitivity to location in the image, rotation (in plane and out of plane, when does it stop working), illumination changes, scale, ... Additional evaluations can be based on the simulated situation (see Section 5).

Summarizing, the steps you should do are:

- Select at least two of the markers/sequences (2a and 2b together do not count as two)
- For each marker you selected
  - Decide on reference points on the marker
  - Develop an approach that can localize (and/or track) the marker in images
  - Make sure you can compute your reference point positions in the images from the information gained in the step before
  - Run you approach on the easy as well as the hard sequence
  - Evaluate the performance of your approach frame by frame and overall (aggregate the frame by frame results)
  - Describe the developed approach in sufficient detail in your report, motivating choices
  - Describe how you evaluated your method and the results in your report

4 Tracking points using the image Jacobian

Before tracking images, you are in this section asked to check your tracking algorithm independently of the feature extraction.

In the workcell associated to this project you will find a frame called *Marker* placed in the center of the marker plate. Given the pose of this frame relative to the camera, you can compute the image coordinates of it (see section 4.9 of the robotics lecture notes). The simulated camera has a resolution of 1024×768 pixels and a focal length of $f = 823$ pixels. Notice that the equations in the robotics notes assumes that the origin of the image is in the center, whereas in image processing the origin is typically the upper left corner.

Associated to the project are three files *MarkerMotionSlow.txt*, *MarkerMotionMedium.txt* and *MarkerMotionFast.txt* containing 6D poses of the marker frame relative to the world frame. The format of these files are:

```
X Y Z Roll Pitch Yaw
```

where X, Y and Z defines the position and Roll, Pitch and Yaw are the RPY angles measured in radians. The time separation between adjacent frames is Δt. To move the marker you need to find the marker frame in the workcell, cast it to a `MovableFrame` and use the `setTransform(const Transform3D<>&, State&)` method on it.

You should use the image Jacobian in the lecture notes (see section 4.9: The Jacobian and visual servoing for a tool mounted camera). The image Jacobian contains a z variable corresponding to the distance along the optical axis between the camera and the point being tracked. You can choose the value $z = 0.5$ meters in all the experiments.

The procedure for testing your implementation is as follows:

- Use the method from section 4.9 to implement a method that allows you to compute a joint displacement $\mathbf{dq}$ from a desired displacement of an image point (du, dv).
- Set the robot in the configuration $\mathbf{q} = \{0, -0.65, 0, 1.80, 0, 0.42, 0\}$ and $\Delta t = 1s$.
- Store the associated pixel position corresponding to the origin of the marker as the target. That is, when tracking you wish to keep the marker origin at that target position in the image.
- Run a loop where you run through the marker poses in the data file. For each entry, compute the displacement (du, dv) between the current position of the marker origin in the image and the target point. Use the

implemented method to compute the corresponding joint displacement $\mathbf{dq}$.

- Update the joint configuration $\mathbf{q}$. If the joint velocities $\frac{\mathbf{dq}}{\Delta t}$ satisfies all the velocity limits, you may just set $\mathbf{q} = \mathbf{q} + \mathbf{dq}$. Otherwise, you must adjust the update to satisfy the velocity limits (which means that you may not be able to completely track the marker point). The velocity limits can be found in the device file of the robot and thereby loaded directly into RobWork by using getVelocityLimits().
- For each step, store the robot joint variables and tool pose. Plot these data sets in your report.
- Choose now different Δt's in the range $0.05s \leq \Delta t \leq 1s$ with decreasing steps of $0.05s$ ($\Delta t = 1s, 0.95s, 0.90s, ...$) until you are unable to track the marker. Plot the maximum tracking error in pixels as function of Δt for the sequence where you were able to track it.

Perform the tests for each of the three data sets in the files *MarkerMotionSlow.txt*, *MarkerMotionMedium.txt* and *MarkerMotionFast.txt*.

Repeat now the above tests, but rather than using a single point from the marker use the 3 points with coordinates $(0.15, 0.15, 0)$, $(-0.15.0.15, 0)$, $(0.15, -0.15, 0)$ in the marker coordinate frame. Notice, that here these points are pre-specified for this section whereas you are asked to choose them for the images. It should be noticed that these points are typically different from the points used for feature extraction and tracking in section 3 and 5 respectively.

5 Combining feature extraction and tracking

In the last two sections you extracted feature points from images and used a theoretical camera to track feature points. In this section the goal is now to combine the two approaches (using a simulated robot vision system).

Select one of the markers presented in Section 3 that you want to use in the simulated environment. The marker files from Section 4 are used again. But now you are asked to place your marker image so that the center of the image coincides with the origin of the marker frame and the u and v axes of the image are aligned with the x- and y-axes of the marker frame respectively.

Integrate the tracking approach developed in Section 4 so that one, two or three of the points delivered from the feature extraction method can be tracked. To show the impact of the feature extraction on the tracking system we expect you to estimate the feature extraction time (average), say τ and to

compensate for the resulting tracking delay. A simple solution is to subtract τ from Δt when computing velocity limit checks. However, you may also suggest a more advanced solution.

For the points you use for tracking, we ask you to plot the size of the tracking error as function over time for values of Δt that you should properly select based on your estimate of τ and your experience with the exercises from Section 4. You should also plot the joint positions scaled wrt. joint limits and joint velocities scaled with respect to velocity limits.

In addition, we ask you to produce some concluding remarks stating how well you believe to have solved the exercise and what could be improved (and how).

6 Formalities

Deadline for the project is December, 18^{th}, 13:00. The project should be delivered through the SDU Assignment feature on Black Board. Mails will not be accepted as hand-in.

The formal requirements for the hand are the following:

- Implementation and verification of working method for computing the marker positions for at least two markers. Clearly describe your method and how you have evaluated the results.
- Implementation and verification of a working method for computing the robot joint variables for tracking one and three image points. Clearly state how you verified that your method is correct.
- Report clearly explaining the algorithms used; the motivation behind the choices and the tests performed. The report is to be between 2000 and 4000 words.
- A zip file containing the source code developed in the project and a plain text-file explaining the content of the files, how to compile and how to run the tests.

The project is to be solved in groups of two persons. Groups are NOT allowed to copy text or code from each other! In case copying is found both groups will fail the course and be reported to the university officials! If code snippets from the exercise solutions on Black Board are used, make sure they are properly referenced in your report.