MSFT-Visual Servoing Practical 1
Point Based Virtual Visual Servoing

Masters in Computer Vision

UNIVERSITE DE BOURGOGNE

Centre Universitaire Condorcet - UB, Le Creusot

13 December 2017

Team Members:
Mohit Kumar Ahuja
GopiKrishna Erabati
Dousai Nayee Muddin Khan

Supervisor:
Dr. Nathan Crombez

Task 1: Image Acquisition

What we do:
Here, we tried to acquire frames of kinect and display the frames in sequence with a frame rate and detect and track the black disk (dots) printed on white paper.

How we do:
The Image acquisition and dot detection is done by the following sequence:

1. We need to start kinect for this we used *MyFreenectDevice* object, as we acquire images we need *vpImage* object to store images.
   1. getvideo, -- Get the frames
   2. vpDisplayX object, -- to display images
   3. Display and flush -- to prepare display and flush display

   The above steps are run in a loop to effectively stream the kinect.
2. To detect dor, we need a *vpDot2* object. And we initially set the display option of dot detection to display dot detection result on image using *setGraphics* method, and then we use *initTracking* to select the initial position of dot.

   And the tracking of dots in sequence of images is done in following sequence in loop:
   1. getvideo, -- Get the frames
   2. display , -- Display frames
   3. track, -- Track the detected dot
   4. flush. -- To flush the display and effectively display image

Why we do:
This task is done to effectively grab the frames of kinect and display them as a video flow and how VISP library can be used to detect and track dots which can be used as a basis for visual servoing. The tracking result of one dot can be seen in Fig. 1.

Figure 1. Result of tracking one dot

Task 2: Initial Pose Computation

What we do:
Here, we tried to compute the initial pose of object ${}^{c}M_{o}$ in the camera frame so that we can use this as a initial pose of virtual robot.

How we do:

The intrinsic parameters of the camera are defined as; $\alpha_u = 535$, $\alpha_v = 535$, $u_0 = 320$, $v_0 = 240$. This is required because camera projection is involved later. And to estimate the pose of the

camera, we select four points and do some processing on them as explained below and add them to *vpPose* to compute pose linearly using Lagrangian method.
Processing of points:
1. We created a vector of size (4) for 4 points of type *vpPoint*.
2. We set the world coordinate frame at the object coordinate frame and and we set world coordinates of each point in meters.
3. We initiated the tracking of all the 4 dots and center of gravity of four dots expressed in image frame is converted to normalised plane using *convertPoint* method of *vpPixelMeterConversion* class.
4. We now set the x and y of each *vpPoint* using above computed pixel values and we add these points to pose to calculate pose of object in camera frame..
5. As we have the world and pixel values of each point, we can compute the pose.

And we display frame as shown in fig.2.

Why we do:
This is done to get the pose of object in camera frame, as it involves camera projection model to get dots in pixel units , the intrinsic parameters of camera are required. So, by using the method described above we can get the rotation and translation of object frame in camera frame, which can be extrinsic parameters of the model.

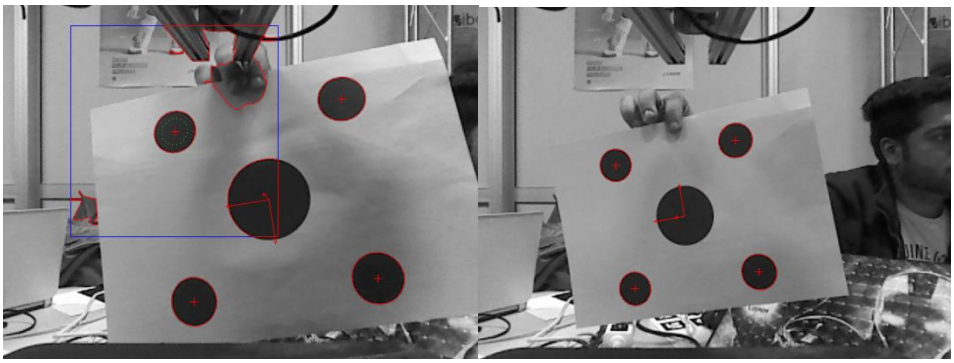

Figure 2. Result of tracking multiple dot's with the computed pose of object in camera frame.

Task 3: Point-Based virtual Visual servoing

What we do:
Here, we try to mimic a robot with camera and set the initial pose of camera and desired pose of camera and we do the processing to get the camera from initial pose to desired pose by visual servoing.

How we do:
In this, we assumed a virtual free flying robot and also set a pose for the free flying robot as ${}^{c}M_{o}$ as we calculated in task 2. We also set the desired pose of robot ${}^{cd}M_{o}$ as below:
`vpHomogeneousMatrix cdMo(0, 0, 0.75, 0, 0, 0); //` We only gave the translation in z-axis.
So as we need to move the robot we need a *vpServo* object and a velocity to move it, computed using control law.
Here we are considering the 4 points as our features. As we already have normalized coordinates of dots tracking we can define four desired features *vpFeaturePoint* using *create*

method of *vpFeatureBuilder* class. As we know the point coordinates in object frame, the features described are the projection of object four points in image, for current pose ${}^{c}M_{o}$. So, we apply frame change on each *vpPoint* and we apply perspective projection using the *projection* method and we update the feature. These feature points are added to the *vpServo* object and control law is computed to get the 6-vector velocity. Then we can update the camera pose using this velocity and we can get the new ${}^{c}M_{o}$ from the robot camera. The above process is run in a loop of iterations so that the initial pose reaches the desired pose.
The complete process of virtual visual servoing can be see in fig. 3 and 4.

The results are uploaded with a short video here: https://youtu.be/WtgNPxDBbuQ
And the code is here: https://github.com/nayeem78/Point-Based-Virtual-Visual-Servoing

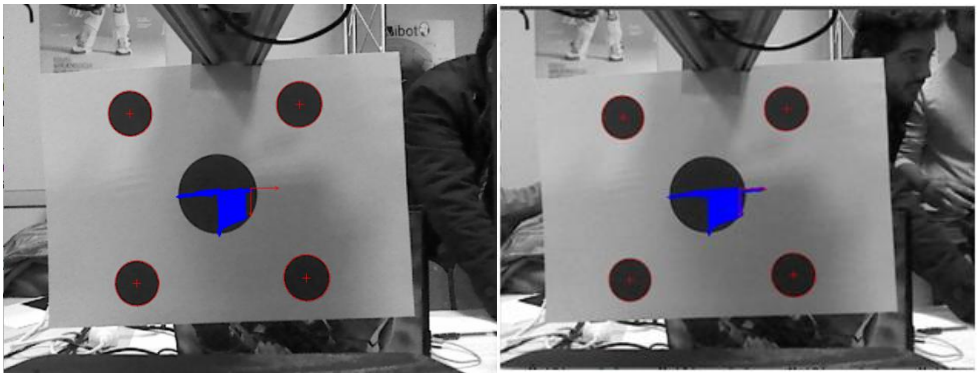

Figure 3.: Virtual robot coming towards the desired location
(Red- Desired Pose, Blue- current Pose merging towards desired Pose)

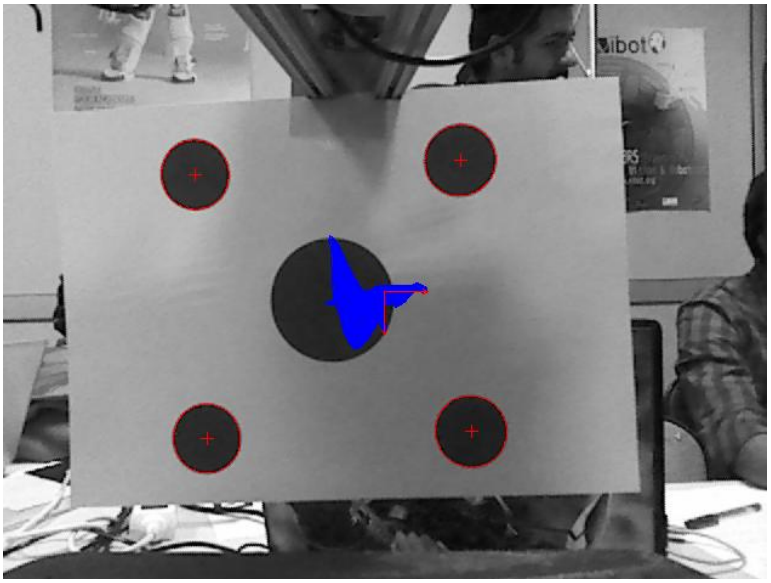

Figure 4. Virtual robot coming towards the desired location with a different initial pose
(Red- Desired Pose, Blue- current Pose merging towards desired Pose)

Conclusion:

We have done dot detection, tracking, computing pose and virtual visual servoing using VISP library. We had seen great advantages of this library as we can track and detect using a single line of code and the data structures defined for holding data are well defined and the class and methods used in the library are well documented with many examples. We found it's easy to use this library for visual servoing in the context of our tasks.