Optical Flow based Visual Odometry for the Aldebaran Nao Humanoid Robot

Madhura Parikh

Abstract

A major challenge for mobile autonomous robots is the accurate estimation of odometry. The accuracy of the odometry sensing determines the success of several crucial techniques like localization. Most mechanical odometry sensors are generally prone to slipping and highly dependent on the terrain - and thus lack the desired robustness. In this project we present a vision based approach to odometry on the robot. For our implementation we use the Nao Aldebaran Humanoid robot. Several approaches have been presented in literature that propose such a system. An interesting challenge on the Nao is that it is a bipedal humanoid. Thus unlike most of the approaches that propose a visual odometry system for wheeled robots, a walking gait is more jerky and result in lower quality images, making the task more challenging. In this project we describe how we implement a visual odometry system based on the concept of optical flow on the Nao, describing our implementation, the challenges we faced and the outcomes.

1 Motivation

Accurate odometry sensing is extremely crucial in mobile robots since it plays an important role in robot localization, especially in GPS denied environments and challenges such as Robocup [13]. Mechanical odometry sensors are very dependent on the terrain, and may fail due to slipping or moving on unknown domains. A more robust approach is to use the vision sensors of the robot to estimate the odometry using information from successive frames since it remains totally unaffected by the noise due to slipping and surface irregularities. Given the availability of cheap and good quality cameras and the increased accuracy that is achievable via a vision-based system, there has been a growing trend to utilize the vision system either partially or fully for odometry estimation. In this project we implement and evaluate an optical flow based visual odometry system on the Nao robot.

2 Problem statement

The problem of odometry estimation is to estimate the change in position over time due to navigation. The change in position is specified as the displacement in the x direction ($\triangle x$), the displacement in the y direction ($\triangle y$) and the change in the heading/rotational displacement θ ($\triangle\theta$) [1]. Unlike traditional odometry sensors that try to measure the wheel-rotation, etc to estimate the position change, visual odometry attempts to solve this problem by analysing successive image frames from a camera sensor. This is highly applicable especially to humanoid/ legged robots, whose jerky motion and absence of wheels makes it challenging to estimate odometry by the conventional approach. Most robots on the other hand are now equipped with cameras and these are one of the most reliable and informative sensors, so a visual approach to odometry is strongly indicated. This is the problem we address in the project. In particular though, we would like the designed visual odometry system to work online in real time on the Nao humanoid

[1] `http://en.wikipedia.org/wiki/Visual_odometry`

for Robocup SPL applications [2]. There are several new challenges that are introduced with this additional constraint.

- Most systems that use a sophisticated computer vision based approach have no strong constraints on the computational resources or no requirement of time efficiency. By contrast we would like the system to run onboard the Nao which has a low end 1.6Ghz Intel Atom processor. We also would like the system to be very fast and efficient since these two elements are crucial for winning in Robocup. Thus while several systems report a real time visual odometry system, the frame rates they achieve in practice are quite low around $1 - 2fps$ whereas for robocup a much higher frame rate of $25 - 30fps$ is a major requirement.
- Several systems that implement visual odometry try to use stereo cameras and a 3D structure from motion approach for visual odometry or panoramic omnidirectional cameras [17]. However the Nao is equipped with a pair of monocular cameras and so our vision system is implemented for monocular rather than stereo vision
- Many of the robot systems [7] are wheeled robots which means that their motion is much smoother as compared to legged robots like the Nao which have a jerkier gait and thus have more noisy and blurred images.

The rest of the report is structured as follows: We describe various contemporary approaches to visual odometry in the next section (section 3). Next in section 4 we describe in detail our implementation of the system on Nao,our hardware and software setup and algorithm sketch. We follow this with a description of some experiments that we used to evaluate our system and the performance and accuracy metrics for the system. In the next section (section 6) we describe some key points of our approach and the lessons we learnt from this project that can be useful for other vision-based projects on the Nao. Finally we conclude with some extensions and improvements to our approach in section 7.

3 Related Work

3.1 Introduction

The paper by Bonin et al [3] provides a good coverage of vision based techniques that have been developed for robot navigation and related tasks. While these span a very broad spectrum ranging from unmanned aerial vehicles to underwater navigation systems, we describe them briefly to set the context for our work since they all are related to the visual odometry task. Next, we take a look at the various vision-based odometry/navigation techniques that have been used for autonomous agents also including a couple that deal with the Nao humanoid. Finally we look at one particular technique - that of optical flow.

As stated in [3] most visual navigation tasks may be bifurcated into two broad categories: map-based navigation and mapless navigation. Map-based techniques require a map of the environment for navigation - these maps must be either pre-provided, while map-building systems use techniques like Simultaneous Localization and Mapping(SLAM) that may be used to build a map as well as localize. Many of these SLAM techniques are based on vision and use input from stereo cameras. Some systems are also based on recovering 3D structures of the environment using algorithms like Structure From Motion(SFM). Mapless systems on the other hand do not require any a priori information of the environment for taking decisions and these may be optical-flow based, appearance based or feature-tracking based. Optical flow based techniques are intuitively based on tracking some features across successive image frames, thereby helping to calculate the apparent motion of these features. These differ from feature-tracking systems in that feature-based techniques typically require an accurate identification of a set of features

- such as edges and landmarks - in the two images that are being compared, to calculate their relative positions. These features are generally detected using various algorithms such as Canny Edge detector, etc. On the other hand optical flow based techniques try to model the movement of pixels, across successive frames, thus not requiring such feature-detection at the 'higher-level'. This is especially advantageous since it can be applied successfully to low quality images that have motion blur and is often used for Unmanned Aerial Vehicles(UAV). Interestingly this technique was in fact adapted from the navigation in bees. Of course optical flow techniques may be optimized by first identifying some features and then computing the optical flow for only those features across the frames rather than for the entire image - accordingly we have the original 'dense optical flow' algorithms [11] and more recent efficient 'sparse optical flow' [4, 27]. Appearance based techniques are based on creating image templates tagged with localization information in a pre-processing stage. While navigating, the robot tries to match the current environment with these stored templates to locate itself by using the previously learned machine classifiers. Clearly these techniques are limited to only well-known environments.

Another interesting technique is that of the visual sonar that was proposed by Velosa et al [18] for fast unknown obstacle avoidance in Robocup with the Aibo robots. The image segmentation stage labelled pixels of all the known colors. In the next step, they used scan-lines emitted radially in the image, at every $5°$ from the robot's center to create a run length of identically colored pixels. These could then be identified as a known object such as a ball, or be treated as an obstacle if they didn't match the criteria for an expected object in the environment. The obstacles of unknown colors were those that caused occlusions of the known-colored floor. This technique could be extended to be used as a visual odometry technique by running it on consecutive image frames to gauge the apparent motion of a known feature/object.

3.2 Visual Odometry : Practical Implementations

Now we examine some visual odometry techniques that have been implemented on actual systems. A good exposition of the progress of visual odometry techniques is available in the tutorials by Scaramuzza et al [24, 9]. It describes use of feature identification algorithms such as Harris corner and Canny edge detectors followed by feature matching algorithms such as SIFT (Scale-invariant feature transform) and model-fitting algorithms like RANSAC (RANdom SAmple Consensus), mostly using stereo vision. The origin of VO was mainly for use in planetary rovers and one of the most successful applications of VO was in the Mars Exploration Rovers(MERS) [20]. They used a similar approach as described above with stereo images. The Visual Odometry helped MERS to impressively navigate unknown and rocky terrains where ordinary odometry gave highly inaccurate results due to slipping or was impossible. While this is quite a different approach as compared to ours it does show that dramatic pay-offs are likely when Visual odometry is used. There are also some other points worth mentioning - firstly since for MERS the main goal was robustness, there the VO system was not tuned for speed and could take orders of seconds for image processing, on the other hand for our Robocup application extremely fast processing is a must. Secondly since their approach was inherently feature based, there were terrains for which no suitable feature could be detected, this scenario required intervention from an earth based operator. Using the optical flow approach, we believe that this dependance on features can be reduced. This is important since no external intervention is allowed in Robocup during an ongoing game.

We next look at the system described in [14] that is designed for highly dynamic and real time applications such as autonomous car driving. They propose using feature tracking combined with an appearance based technique to maximize the robustness of the identified features in a dynamic environment. They also use a Kalman filter to account for the moving state of the system itself. Since they use an appearance based step, they also require classification for which they use Random Forests - which are faster than the typically used Support Vector Machines (SVMs) - and an adaptive bucketing step which improves the classifier accuracy. Though no direct comparisons are given, they report success over existing approaches also showing that their vehicle could successfully stop at a traffic signal on a busy road. One major divergence of their approach from the one we propose is that the use of an appearance based system would require prior hand-labelled training data for the classifier while optical flow does not require any such step. However one encouraging takeaway for us from this approach is that using VO online and in real time is indeed achievable.

3.3 Visual Odometry on the Nao Humanoid

Now we turn to some Visual Odometry techniques that have been specifically designed for the Nao humanoids. The paper by Nadarajah et al [22] provides a concise summary of various vision based techniques that teams have used in Robocup. It mentions some approaches that have used optical flow mainly for obstacle avoidance, though with a very low frame rate - which is something we may have to watch out for. Though it does not mention any team that has used VO, there has been some recent work in this area. For instance in [23] the authors give a good description of the challenges for VO on a humanoid and introduce an approach for a VO approach that is robust to motion blur. According to them, unlike wheeled robots, walking, turning and other such motions on the Nao may result in undesirable blurring of images that can be a failure point for various feature tracking methods. To tackle this, they propose a pre-processing step for every image, prior to the feature extraction. For this they estimate the Point Spread Function(PSF), however since the blur will not be uniform for the entire image, they first cluster the image into smaller areas and compute local PSFs for each image area.Next they propose a variation of the SIFT scheme for feature extraction. For matching features between frames they use the five point algorithm, obtaining the five points using a Best Bin First algorithm. They report that their feature detection is very robust to motion blur unlike SURF(Speeded Up Robust Features) and SIFT methods that can detect very few features when given the same noisy images. However they report $1s$ to process a $640x480$ image on a 2 Ghz 2 core PC. Clearly this work introduces a number of points, we should watch out for - first it may difficult to perform VO given noisy images, which are certainly expected for an application like robot soccer, secondly dealing with these images may introduce additional time costs which we cannot afford.

A more recent work that uses VO for the Nao humanoid is [8]. Their aim is to develop a more accurate odometry to improve localization. Here they develop a system that is independent of any particular hardware by using the Robotics Service Bus(RSB) as middleware. Also rather than pre defining features that must be tracked in each frame, they use Structure From Motion(SFM) to detect the likely features in each frame. This ensures that their system is also independent of a particular environment. To deal with blurred images they use a combination of both conventional and visual odometry and return the final result based on inputs from both of these. In case of noisy images, if the VO fails to work, then only the input from the conventional odometry, using a weighted mean method. The final result is considerably more accurate as compared to the one obtained from ordinary odometry alone. On a 64 bit core $i5$ laptop the frame rate that they could achieve was $8fps$. While for the Robocup application we do not require the system to be independent of the environment, an interesting idea to follow up would be their fusion scheme that can help to gracefully degrade when the VO fails to work for very noisy images. Yet another paper that describes VO for Nao is given by the Dutch team [16]. However currently, this is still a work in progress and reliable results are yet to be reported. Similar to the previous work, they use FAST features and the FLANN(Fast Library for Approximate Nearest Neighbors) for feature matching. The paper also provides a good layout of the various algorithms that are typically applied at different VO stages. Most of these works also mention using the implementations available in the OpenCV library [5] for several key algorithms.

3.4 A look at optical flow approaches

Finally we delve into how optical flow has been used for autonomous robots. The paper by Thrun et al [19] is a novel application of optical flow though for a different purpose than VO. The problem they address that most sensors can work only in the short range. They propose a self-learning approach in which they detect an feature in the current image and trace it back to a corresponding image frame taken some time in the past by maintaining a chain of optical flow vectors to that previous frame. By this kind of "reverse optical flow" they can map the short range characteristics of the image features to their long range characteristics, effectively improving the sensing range of the robot. While the rest of the paper mainly deals with applications of this newly acquired sensing skill, there are several important specifics that are mentioned for optical flow which may be potentially useful for us. First to maintain an array of optical flow vectors, they describe an efficient memory storage scheme. Also the algorithms they found most effective are the Shi-Tomasi algorithm for feature extraction and the Lucas-Kanade tracker for tracking these across frames - these are useful pointers relevant to our work.

We would like to conclude with the mention of the work by Campbell et al, that provides some evaluation criteria for VO systems also showing the implementation for a simple VO system which is based on the optical flow algorithm based on some simplified assumptions. They also describe how this system performs with regard various criteria and thus can be a useful reference as we design our system.

4 Our approach and implementation

In this section we first give a brief overview of our hardware configuration, our software setup and finally describe our implementation in detail.

4.1 Hardware configuration

As we mentioned previously, we designed our system for the Nao Aldebaran Humanoid robots that are used in the Robocup Standard Platform League [2]. The Nao is equipped with two monocular cameras with a frame rate of $30fps$. One of these is near the forehead (top camera) and streams distant objects (horizon) while the other (bottom camera) is located above the mouth and streams the close-by surroundings. We use live streams captured from both these cameras for our task. The Nao also has a linux kernel and a low end Intel Atom $1.6GHz$ processor that allows for onboard processing and fully autonomous behavior.

4.2 Software setup

For developing our code, we started with the UT Austin code base and used the UT Nao Tool developed here at UT, since it provides a convenient and modular approach for programming on the Nao. We also used the OpenCV library for some standard computer vision algorithms which we describe later. Since it is possible to communicate to Nao via a wireless or ethernet connection, we used the $64-bit$ Ubuntu OS for developing and debugging our code and then deployed it to Nao for actual testing and running. We used git and a private GitHub repository for version control and managing the code.

4.3 Algorithms

For framing our algorithm we were mainly guided by the paper by Campbell et al [7]. This is a paper that tried to meet similar goals to ours - with one major difference - their system was

[2]`https://community.aldebaran-robotics.com/nao/`

designed for a wheeled rather than a bipedal robot. In the subsequent discussions that follow we will also discuss the implications of this.

The gist of our algorithm whose pseudo code appears in table 1 is as follows: We first take the image frame being currently streamed by our Nao. Then we use the Shi Tomasi corner detection algorithm [25] to 'filter out' the most promising features whose movement we can track from the current frame to the successive frame. Next we use the pyramidal extension of the Lucas Kanade algorithm [4] to calculate the optical flow vector in moving from the current frame to the successive frame. Broadly the optical flow algorithm takes as input the features shortlisted by the Shi Tomasi detector in the current frame and returns the corresponding matches (if it can find them) in the next frame. More details of these algorithms are elaborated below where we discuss the corresponding functions available in OpenCV.

Next for all the points in the first frame, which we were able to locate in the next frame, we compute the optical flow vector. Here the features that the Shi Tomasi detector returns are just the pixels (u, v) in the image which are corners. Thus if the feature in the original frame was (u_p, v_p) and the Lucas Kanade algorithm locates this to be at (u_n, v_n) in the next frame, then the flow vector for this feature is:

$$(\Delta u, \Delta v) = (u_n - u_p, v_n - v_p) \tag{1}$$

We use the flow vector as computed in (1) to then obtain the *ego-motion* of the robot. The ego-motion of the robot is returned as the translational and rotational displacement $(\Delta x, \Delta y, \Delta\theta)$ in the ground plane using the optic flow information that we computed in the image frame. To facilitate this translation from the image frame to the world frame, we assume the pinhole model for the camera [3]. Similar to [7] we used the bottom camera images (which record the immediate surroundings and the ground plane of the robot) for calculating the translational displacement. On the other hand we used the top camera images (which record distant objects and may be considered to be on the horizon) for calculating the rotational displacement. This is because they would show less effects of translation due to the robot's egomotion. For each of these two sets of images, we used two consecutive image frames, calculating the optical flow vector from the latest frame to its previous frame.

Now to translate the motion observed in the image frame to the word coordinate frames, we used the standard calibration parameters of the camera and basic trigonometry to obtain both the rotational and translational displacement in the world coordinate frame. For translation, we directly used the `CameraMatrix::getWorldPosition` available in the Austin Villa source code to project our image coordinates to the ground plane. So for instance let (u_{t-1}, v_{t-1}) be the coordinates of a corner in the first frame and let (u_t, v_t) be its match in the next consecutive frame. Also assume that our camera has focus fx and let the the principal point be given as (cx, cy). With these variables defined, we use the following equation to estimate the rotational displacement of the robot:

$$\begin{array}{lcl} \theta_{t-1} & = & \arctan(\frac{u_{t-1}-cx}{fx}) \\ \theta_t & = & \arctan(\frac{u_t-cx}{fx}) \\ \Delta\theta & = & \theta_t - \theta_{t-1} \end{array} \tag{2}$$

To calculate the translation we project both (u_t, v_t), (u_{t-1}, v_{t-1}) to the ground plane using the `CamerMatrix::getWorldPosition` already available in the Austin Villa source code. Before doing this coordinate transform however, we first remove the rotational effect that was calculated in the previous equation so more accurate estimates of the displacement cna be obtained. To do this we multiply the (u, v) coordinates in the image frame by the following matrix - which follows from basic coordinate geometry - to obtain their rotation free counterparts (u', v').

[3] `http://en.wikipedia.org/wiki/Camera_resectioning`

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \tag{3}$$

Since there are several optical flow vectors in the image, we thus obtain several values for the rotational displacements as well as the translational displacements. To obtain the net displacements, we simply return the median value for each of $\Delta x, \Delta y$ and $\Delta\theta$. We experimented with other summary statistics, like returning the mean of the values or returning the mean of the maximum k values. But we found that the best estimates were returned by using the median.

Algorithm 1 Visual Odometry Algorithm

```
1: $cummulative_{\Delta x} \leftarrow 0$
2: $cummulative_{\Delta y} \leftarrow 0$
3: $cummulative_{\Delta \theta} \leftarrow 0$
4: while True do
5:     $prevGray \leftarrow$ grayscale image from last frame
6:     $curGray \leftarrow$ grayscale image from current frame
7:     $features \leftarrow ShiTomasiDetector(prevGray)$
8:     $trackedFeatures \leftarrow LucasKanadeOpticalFlow(prevGray, curGray, features)$
9:     $i \leftarrow 0$
10:    while $i < trackedFeatures.length$ do
11:        $i \leftarrow i + 1$
12:        if $trackedFeatures[i]$ is valid then
13:            $\Delta\theta_i \leftarrow CalculateRotation(trackedFeatures[i], features[i])$
14:            $angles.add(\Delta\theta_i)$
15:            RemoveRotation($trackedFeatures[i]$)
16:            RemoveRotation($features[i]$)
17:            ProjectOnGround($trackedFeatures[i]$)
18:            ProjectOnGround($features[i]$)
19:            $(\Delta x_i, \Delta y_i) \leftarrow CalculateDisplacement(trackedFeatures[i], features[i])$
20:            $xDisplacements.add(\Delta x_i)$
21:            $yDisplacements.add(\Delta y_i)$
22:        end if
23:    end while
24:    $average_{\Delta x} \leftarrow median(xDisplacements)$
25:    $average_{\Delta y} \leftarrow median(yDisplacements)$
26:    $average_{\Delta \theta} \leftarrow median(angles)$
27:    $cummulative_{\Delta x} \leftarrow cummulative_{\Delta x} + average_{\Delta x}$
28:    $cummulative_{\Delta y} \leftarrow cummulative_{\Delta y} + average_{\Delta y}$
29:    $cummulative_{\Delta \theta} \leftarrow cummulative_{\Delta \theta} + average_{\Delta \theta}$
30: end while
```

4.4 A look at OpenCV functions and parameters

OpenCV [5] is an open source $C++$ library that offers efficient implementations of a host of state-of-art computer vision algorithms. We leverage off the implementations already available in this library for the Shi Tomasi detector as well as the pyramidal Lucas Kanade algorithm. Here we discuss the various parameters that we used for these functions and their significance and meaning. (We used OpenCV version 2.4.0)

The Shi Tomasi feature detector appears as the `cv.goodFeaturesToTrack` function in the OpenCV :

```
    void goodFeaturesToTrack(InputArray image, OutputArray corners, int maxCorners, ↩
        double qualityLevel, double minDistance)
```

There are several parameters that this function takes, but we mention only the ones we used and the ones that were relevant to us.

- `image` is the original image frame converted to grayscale.
- `corners` This array is filled in by the function, with the promising points(corners) it shortlists in the input image
- `maxCorners` This is the maximum number of the promising features we want the function to return.
- `qualityLevel` This parameter determines which points we would like to retain based on their quality. The quality is determined by using the eigenvalues of the system model used in the algorithm. The corner that has the minimal eigenvalue in this model is considered the best - and all corners whose quality is less than this minimal eigenvalue multiplied by `qualityLevel` are rejected. Thus this parameter helps us control how many features the detector will find. By setting the quality level to be high, we may get very few points that are more reliable, whereas in the opposite case, we get a large number of corners but with less promise of reliability. We tried several values for this parameter. In the extreme case we tried both 0.005 - to get back a large number of corners - for our setup on an average this would return around (of course also based on the `minDistance` below) 80 points and also 0.1 which generally returned only around 4 points.
- `minDistance` This specifies the maximum Euclidean distance within which we would like to find the point in the next frame from its original position in the current frame. Again by setting the value to a large value, we allow for a greater displacement between frames and likely more spurious vectors whereas a smaller value would allow for very small displacements. We started conservatively by setting this to 20 which was all right for purely translational motion but later increased it to 100 since this was more realistic when there were also rotations. Overall though we found that the `qualityLevel` was much more critical in deciding the corners that would be detected.

Other parameters to this function included specifying the portion of the image we wanted to detect the corners rather than the entire image and using the Harris detector in place of the Shi Tomasi. However we did not use these options and so do not elaborate on them here. A major limitation of this function is that it is the bottleneck for the entire algorithm. It consumes the maximum amount of time in the entire algorithm. With very liberal values set for the `qualityLevel`, `minDistance` and `maxCorners`, $(0.005, 20$ and 4 respectively) the function still takes around $0.1s$ per image frame, when run on the Nao - a factor of 100 times more than the next most computationally intensive operation - calculating the pyramidal Lucas Kanade optic flow. Clearly if the visual odometry system is to be successfully deployed for time critical applications like Robocup - this is one of the first areas to improve.

In addition to the Shi Tomasi detector, OpenCV also has one more function `cv.cornerSubPix` - this helps to pinpoint the position of the corners to sub-pixel level accuracy by iterating through a window surrounding the corner. This function looks as follows:

```
void cornerSubPix(InputArray image, InputOutputArray corners, Size winSize, Size ↩
    zeroZone, TermCriteria criteria)
```

- Again `image` has the same meaning as above.

- The `corners` is the filled up by the Shi Tomasi detector with the initial corners and improved by this function.
- `winSize` indicates the box around the pixel-level corner in which we would like to search for the subpixel level refinement. We set the window to be $(3, 3)$.
- The one other relevant parameter is the `criteria`- this specifies that the search should stop either when a certain number of iterations have been completed or when the subpixel coordinates are steady within some limit. We set the value of this to `(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 20, 0.3)` i.e the process stops after either 20 iterations or when the center moves by less than 0.3 units in the subsequent iteration, whichever is fulfilled first. While we haven't carefully tried to tune the parameters to this function, we believe that the current values yield us satisfactory results. Besides the function is very fast as compared to the Shi Tomasi detector - so the more refined corner positions can be obtained, practically free of cost.

The final OpenCV function we use is the pyramidal Lucas Kanade algorithm for calculating the optical flow. As explained in [4] the idea of using pyramids improves the original Lucas Kanade algorithm by making it more 'robust' to larger displacements while at the same time maintaining the accuracy. Roughly this creates variuos levels of the image - so for instance for a 640×480 image the image at level 4 would be a downsampled image with resolution 40×30, at level 3 we have a 80×60 image, at level 2 we have a 160×120 image, at level 1 we have a 320×240 image and finally at the level 0 we have our original 640×480 image. The algorithm proceeds by calculating the optical flow first at the highest pyramid level, eg level 4 and then recursively calculating the flow in images at the lower levels. This helps it to robustly deal with displacements that are visible only on a larger scale. As is evident from here, a 4 level pyramid seems most suitable for a 640×480 image, and this the value we use in the corresponding OpenCV function:

```
void calcOpticalFlowPyrLK(InputArray prevImg, InputArray nextImg, InputArray prevPts, ↩
    InputOutputArray nextPts, OutputArray status, OutputArray err, Size winSize, int ↩
    maxLevel, TermCriteria criteria)
```

Here again we explain the most important parameters:

- `prevImg` and `nextImg` are the grayscale versions of the current and the next image frames.
- `prevPts` are the points we detected from the Shi Tomasi feature detector.
- The `nextPts` is filled up by the optical flow algorithm with the corresponding points matched by it in the next frame.
- `err` is the approximated error for the matched points, returned by the algorithm (only if it was able to locate them).
- `status` is a vector which is set to 1 at a particular index if the feature at the corresponding index in `prevPts` was found.
- The `winSize` and `criteria` are similar to what we described in the `cornerSubPix` function and we set them to the same value here as well.
- Finally `maxLevel` is the number of pyramid levels, we would like for our image. As we discussed earlier for our 640×480 image, 4 levels is probably the best choice.

5 Experiments and evaluation

To evaluate our system, we ran several tests on the Nao humanoid robot. We wanted to evaluate the system for both its translational and rotational odometry estimation. First we enlist the different experiments we performed. Next we discuss the reasons for the results we obtained.

Rotation Test 1 In this we made the Nao stand still and do a head scan. We ran the algorithm to obtain its estimated values at different points during the scanning. For our ground truth we used the output of the head yaw as sensed by the proprioception. The results we obtained are displayed in table 1 below. Note that in this case, the robot made two sweeps and thus the angles are presented in an ascending order rather than in the actual order that they were measured.

Rotation Test 2 In this experiment we made the robot to turn continuously with its feet. We set its translational velocity to 0 and the turn angle to 10°. Again we compared the angle outputted by the VO system with the ground truth - in this case the ground truth was taken to be the angle as manually eyeballed by an observer at that instant. Thus in this case the ground truth values are appproximate rather than exact. The results we obtained appear in table 2

Translation Test 1 In this test we made the robot walk straight in the positive x direction and compared the VO output with manually measured values at various instants. Unfortunately the system behaved very unpredictabely in this case. We found that the values we measured were quite off their actual values. The one optimistic fact in this case was that the system could distinguish between no motion and motion in the positive or negative direction.

Translation Test 2 Here we put the Nao atop a wheeled carrier and measured the odometry estimates comparing them against the manual measurements. However we obtained the same sort of erratic results as in the previous case.

Table 1: Rotation Test 1

Visual Odometry output(degrees)	Proprioception output(degees)	Error(degrees)
52.8	44.9	-7.9
55.9	54.4	-1.5
54.4	64.3	9.9
64.2	68.2	4
64.0	75.7	11.7
70.2	77.6	7.4
64.0	86.2	22.2
67.3	89.5	22.2

Table 2: Rotation Test 2

Visual Odometry output(degrees)	Manual Approximation	Error(degrees)
29	45	16
79	90	11
104	135	31
146	180	34
281	360	79

In both the rotation tests, though the values measured are not very accurate, the system was able to distinguish when the head was moved in the positive direction $[0, \frac{\pi}{2}]$ and also the deflection point, when the head was moved in the negative direction $[0, -\frac{\pi}{2}]$. Below we include a plot of the angular errors when plotted against the true angles. The errors increase as the angles become larger, which may be because larger displacements are generally a failure case for optical flow algorithms.

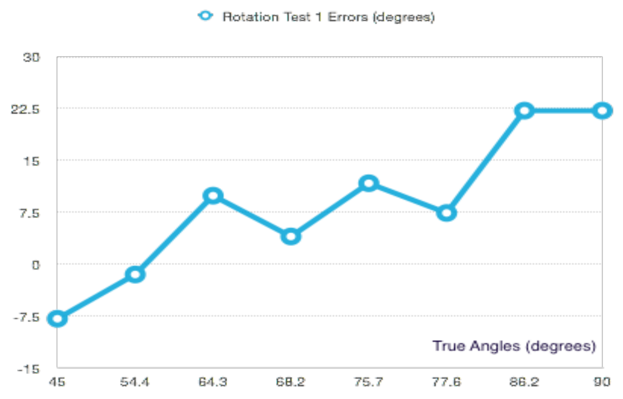

Figure 1: Error vs true angle for Rotation test 1

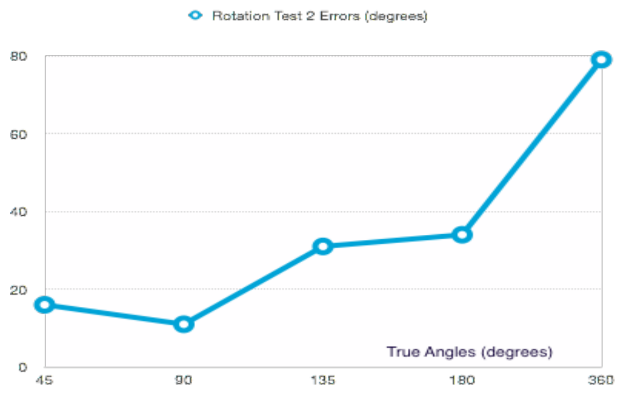

Figure 2: Error vs true angle for Rotation Test 2

Timing requirements

The algorithm is computationally very expensive when considered with respect to various time-critical applications like Robot soccer. For instance with the Shi Tomasi feature detector and the Lucas Kanade optical flow algorithm, the speed ranged from $0.8 - 2fps$. Profiling the code revealed that the culprit was the Shi Tomasi detector. When we used a different set of features, such as *FAST features*, we were able to improve the frame rate to some extent. Here the frame rate rose to around $5 - 6fps$. In this case the bottleneck was not the feature detector but rather the Lucas Kanade optical flow algorithm. We believe that the reason for this is that FAST typically returns a larger number of features as compared to Shi Tomasi - and this increase in the number of features led to a slow down for the Lucas Kanade filter. When we ran the FAST feature detection algorithm independently, there was a dramatic rise in the frame rate - it jumped to $25 - 29fps$.

As it can be seen the system performed poorly on most tests and did not have an impressive performance on the timing front either. The one case in which it showed at least some coherence was in rotation tests. We believe that the reason for this is that the disturbance due to the legged gait of the robot was entirely absent. A similar thing could not happen for the Traslation Test

2 because the wheeled motion was not the intrinsic motion of the robot. Thus the push-pull forces that were exerted onto the carrier likely disrupted the flow field of the robot. With the background of these poor results we would now like to move to the next two sections. In the next section we discuss some limitations of our approach and how viable it really is. In the section following that we talk about future directions of this project that could possibly lead to more promising results.

6 Keypoints and lessons learnt

What are some other limitations of the current system? One major limitation of the current system is that it makes no distinction amongst the various flow fields that it sees. For instance several flow vectors may be due to moving objects in the environment rather than caused by the robot's ego-motion. Since no effort is made to filter out these extraneous vectors, the end result will not be that accurate.

Another major limitation of this system is that the robot for which it is implemented is bipedal rather than wheeled. Most systems that have successfully used visual odometry have been for wheeled robots. The motion due to wheels is very smooth and the smoothness leads to infact one criteria on which we can choose the likely flow vectors that represent the ego-motion of the robot. Campbell et al [7] mention such a criteria in their approach. To obtain the smoothness of a vector, they keep track of vectors upto 3 frames ahead and upto 7 frames backward and calculate the angular displacement of the vector. If the vector shows too large an angular displacement across this history of frames - it is considered 'unsmooth' and discarded. We tried similarly to keep track of vectors across frames. However we found that for our case most vectors seemed to lose relevance after 2 frames and the best results were obtained when the features were tracked from frame to frame rather than by maintaining a history. For legged bipedal robots lke the Nao, the motion is discontinuous and jerky. Thus to design a VO system on such systems, a prerequisite would be to first derive a model of the noise caused by the legged motion to the optical flow. While we were unable to make much progress in this direction we believe that having such a model in hand should lead to much better results.

Is optical flow a viable approach for VO in time critical applications? Based on our experience of designing the system for VO, we are strongly inclined to believe that optical flow does not seem to be a good approach for fast-paced and demanding challenges like Robocup. While it is likely that the results should improve, if we are able to successfully filter out the right optical flow vectors and model the noise due to the legged gait - time still remains a major obstacle. For instance the dutch Nao team [16] designed such a system but were unable to meet with much success. Most systems that have implemented optical flow based algorithms have been limited by their frame rate, reaching in the best case a frame rate of around $10fps$. However in the general case most real time systems have not been able to exceed a frame rate of $1-2fps$.

Because of the nature of the optical flow algorithm, it cannot deal with very large displacements. This was somewhat offset by using pyramidal optical flow, but large displacements still remain a challenge for optical flow algorithms. Because of this we feel that optical flow algorithms are more suited for tasks like obstacle avoidance, where it is sufficient to detect a change in the flow vectors, but a precise measurement is not required as in case odometry applications.

Another major challenge to optical flow is the ability to distinguish flow vectors due to different types of motions and to correctly identify which flow vectors are due to the robot's egomotion. In [12] they use a clustering approach to classify similar optical flow vectors into different groups and then label each group with the motion it is likely to represent. However we believe that such an approach can be very time consuming.

A recent new approach to calculating the optical flow was proposed in [27] where they focused on local optimization rather than on global optimization to decrease the timing requirements. While the algorithm is available in OpenCV there are several constraints on the form of inputs it can accept. However we believe that if this algorithm was implemented and used with a speedy feature detection algorithm like FAST, it would be interesting to look at the results.

7 Where do we go from here?

While the results of this project were not too encouraging, we believe that there are several other approaches that can lead to a more successful visual odometry system. We have also come to the conclusion that such systems would be based on exploiting peculiarities of a particular domain such as Robocup to give them the necessary thrust for speed and accuracy. Before we describe a couple of such approaches, we also describe some pipelines of general purpose algorithms that could serve our purpose better. For feature detection, we believe that rather than using Shi Tomasi corner detection, a much more lightweight extractor such as FAST features should be used. While the features may be much less reliable than the ones detected via Shi Tomasi, a post-processing step like RANSAC could help to focus only on the inliers. For matching the points across consecutive frames, an algorithm like FLANN [21] or PatchMatch [1] could be used. Infact the FAST $\rightarrow$RANSAC $\rightarrow$ FLANN chain has been used by Aldebaran themselves to design a visual compass for Nao [4]. Both FLANN and PatchMatch can very efficiently find matching points between successive frames.

However to really bring about a dramatic change in the time and speed - we may need to change the entire approach altogether. For instance rather than going with a feature tracking approach it may be more promising to go with an appearance based approach. Here we describe three such successful applications of appearance based approach. A very interesting appearance based approach was first proposed by Sturm et al [26] for a visual compass application. For their features they use the transition frequency between pairs of color classes (i.e bigrams of colors) on vertical scan lines of a segmented image. Initially they learn a cylindrical map following a histogram distribution for all possible yaw angles. They then use maximum likelihood to predict their current angle based on the feature vector they extract and the previously learned cylindrical map. They mainly used this to improve the quality of their localization. Moreover they show that on the Sony Aibos(the previous brand of robots used in the Robocup SPL) their approach could support the full frame rate of $30fps$. One draw back of their approach was that since the cylindrical map was learnt only once, at the beginning , the predicted angles would start showing some drift from the actual value as the robot moved away from this origin. This drawback was corrected in the Dutch Nao team's implementation ViCToria [15] on Nao, where they extend this approach and further allow updates to the map to occur each time the robot comes to a new part of the field, by dividing the field into 8 grids with $180°$ angle bins in each grid. In [10], rather than using color bigrams as features, the Robocup team from the University of New South Wales, runSWIFT, proposes to use extremely fast $1D$ SURF features. All these methods have been very efficient and fast suggesting that such appearance based techniques maybe more feasible than the feature tracking approach of our optical flow algorithm.

[4]`http://www.aldebaran-robotics.com/documentation/naoqi/vision/alvisualcompass.html`

8 Conclusion

In this project we used an optical flow based approach to implement a Visual Odometry system on the Nao humanoid. From our experiments we showed that as it currently stands, the system would require a major overhaul to be useful - specifically it is necessary to model the effect of the discontinuous bipedal robot walk on the optical flow vectors, to get better odometry estimates. The system also needs to be optimized for speed, however as we discussed from contemporary literature, there may be an upper bound on the amount of improvement that may be introduced. Thus it may be necessary to switch to an entirely new approach such as an appearance based approach which has shown some success in the past.

9 Acknowledgements

I would like to thank Professor Stone for encouraging me to explore a challenging and new area for my class project. I am especially grateful to our class TA Jacob Menashe for patiently answering many questions and for his expert advice on troubleshooting programming issues we ran into. I am also indebted to my parents for their unconditional support and love at all times and to all my friends for their kindness and help.

References

[1] Connelly Barnes et al. “PatchMatch: a randomized correspondence algorithm for structural image editing”. In: *ACM Transactions on Graphics-TOG* 28.3 (2009), p. 24.

[2] Samuel Barrett et al. “UT Austin Villa 2012: Standard Platform League World Champions”. In: *RoboCup-2012: Robot Soccer World Cup XVI.* Ed. by Xiaoping Chen et al. Lecture Notes in Artificial Intelligence. Springer Verlag, 2013. URL: `http://www.cs.utexas.edu/users/ai-lab/?LNAI12-Barrett`.

[3] Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. “Visual navigation for mobile robots: A survey”. In: *Journal of intelligent and robotic systems* 53.3 (2008), pp. 263–296.

[4] Jean-Yves Bouguet. “Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm”. In: *Intel Corporation* 5 (2001).

[5] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* O’reilly, 2008.

[6] Jason Campbell, Rahul Sukthankar, and Illah Nourbakhsh. “Techniques for evaluating optical flow for visual odometry in extreme terrain”. In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on.* Vol. 4. IEEE. 2004, pp. 3704–3711.

[7] Jason Campbell et al. “A robust visual ‘odometry and precipice detection system using consumer-grade monocular vision”. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on.* IEEE. 2005, pp. 3421–3427.

[8] Šimon Fojtuu, Michal Havlena, and Tomáš Pajdla. “Nao robot localization and navigation using fusion of odometry and visual sensor data”. In: *Intelligent Robotics and Applications.* Springer, 2012, pp. 427–438.

[9] Friedrich Fraundorfer and Davide Scaramuzza. “Visual Odometry: Part II: Matching, Robustness, Optimization, and Applications”. In: *Robotics & Automation Magazine, IEEE* 19.2 (2012), pp. 78–90.

[10] Sean Harris et al. “Robocup Standard Platform League-rUNSWift 2012 Innovations”. In: *Proceedings of the 2012 Australasian Conference on Robotics & Automation (ACRA).* 2012.

[11] Berthold KP Horn and Brian G Schunck. “Determining optical flow”. In: *Artificial intelligence* 17.1 (1981), pp. 185–203.

[12] Tae-Koo Kang, Hee-Jun Song, and Gwi-Tae Park. “Environment Recognition System for Biped Robot Walking Using Vision Based Sensor Fusion”. In: ().

[13] Hiroaki Kitano et al. “Robocup: The robot world cup initiative”. In: *Proceedings of the first international conference on Autonomous agents.* ACM. 1997, pp. 340–347.

[14] Bernd Kitt, Frank Moosmann, and Christoph Stiller. “Moving on to dynamic environments: Visual odometry using feature classification”. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on.* IEEE. 2010, pp. 5551–5556.

[15] Patrick M de Kok, Georgios Methenitis, and Sander Nugteren. “ViCTOriA: Visual Compass To Orientate Accurately”. In: ().

[16] C Kooijman et al. “NAVIGATE-Nao Visual Gait and Trajectory Estimation”. In: *Project report, Universiteit van Amsterdam (February 2013)* (2013).

[17] Frédéric Labrosse. “Visual compass”. In: *Proceedings of Towards Autonomous Robotic Systems, University of Essex, Colchester, UK* (2004), pp. 85–92.

[18] Scott Lenser and Manuela Veloso. “Visual sonar: Fast obstacle avoidance using monocular vision”. In: *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on.* Vol. 1. IEEE. 2003, pp. 886–891.

[19] Andrew Lookingbill, David Lieb, and Sebastian Thrun. “Optical flow approaches for self-supervised learning in autonomous mobile robot navigation”. In: *Autonomous Navigation in Dynamic Environments.* Springer, 2007, pp. 29–44.

[20] Mark Maimone, Yang Cheng, and Larry Matthies. “Two years of visual odometry on the mars exploration rovers”. In: *Journal of Field Robotics* 24.3 (2007), pp. 169–186.

[21] Marius Muja and David G Lowe. *Flann, fast library for approximate nearest neighbors.* 2009.

[22] Sivadev Nadarajah and Kenneth Sundaraj. “Vision in robot soccer: a review”. In: *Artificial Intelligence Review* (2013), pp. 1–23.

[23] Alberto Pretto et al. “A visual odometry framework robust to motion blur”. In: *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on.* IEEE. 2009, pp. 2250–2257.

[24] Davide Scaramuzza and Friedrich Fraundorfer. “Visual odometry [tutorial]”. In: *Robotics & Automation Magazine, IEEE* 18.4 (2011), pp. 80–92.

[25] Jianbo Shi and Carlo Tomasi. “Good features to track”. In: *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR’94., 1994 IEEE Computer Society Conference on.* IEEE. 1994, pp. 593–600.

[26] Jürgen Sturm and Arnoud Visser. “An appearance-based visual compass for mobile robots”. In: *Robotics and Autonomous Systems* 57.5 (2009), pp. 536–545.

[27] Michael Tao et al. “SimpleFlow: A Non-iterative, Sublinear Optical Flow Algorithm”. In: *Computer Graphics Forum.* Vol. 31. 2pt1. Wiley Online Library. 2012, pp. 345–353.