Visual Object Recognition and Tracking System

Final Report

Kevin Briggman

Thomas Lorimor

Advisor: Frank Merat

April 16, 1999

EXECUTIVE SUMMARY

Reliable object recognition and object tracking are of great importance to the design of intelligent systems. Several methods have been proposed, but the Visual Object Recognition and Tracking System concentrates on dynamic template matching with deformations. A target object is tracked using a Connectix Quickcam by matching the colors seen in the object with all possible screen locations. In addition to color recognition, information is derived about the horizontal position and the distance of the object from the camera. Images are processed on a single board computer and the position relative to the tracking robot is sent to a robot controller. The robot controller makes decisions on how to move and turns the decisions into motor commands to drive the robot. The camera, image processor and robot controller are all self-contained on the robot which results in an autonomous implementation. Any behavior can be programmed onto the robot controller such as line following, obstacle avoidance, playing a game of tag, etc. The system successfully tracks a drone robot driving a random pattern and maintains a constant distance at all times.

The system has been designed to be completely scaleable. The tracking algorithm can process data at resolutions up to 640 x 480 and still deliver a decent frame rate. Performance is directly tied to processing power and any increase in processing ability results in increased performance. Increased performance here is defined as more accurate object recognition and better tracking of fast motions. The robot response is also closely tied to processing speed, but indirectly. Faster processing between frames leads to a higher frame rate. High frame rates enable the robot to respond more accurately to small and fast movements. The hardware platform described here performs reasonably well given the limited processing ability.

INTRODUCTION

Computer vision is a vast field of research with many applications. Of particular interest to this project are attempts to recognize and track targets in a video image stream. There are several methods that have been proposed to achieve reliable performance. Some include edge detection, implementation and training of neural networks, and template matching. A neural network is a powerful way to store and recall data such as the position of a target, but it is computationally intensive. It limits the amount of input data (specifically pixels in the image processing case) that can be considered in a reasonable amount of time. An alternate solution is to perform some sort of template matching. It involves searching for an image of a feature of sub-picture that is already known. This can be performed at any resolution and can yield good results when slight modifications to the basic algorithm are made.

Simulation of tracking algorithms is a useful way to demonstrate the ideal capabilities of a solution. However, it is more impressive to implement the algorithm on a hardware platform and test it under real world conditions. An ideal platform has excellent processing ability and yet is compact enough to fit on a mobile vehicle. The solution to the visual tracking problem should not be limited to passive observation. A complete system must be able to react with its surroundings intelligently.

The complete problem, then, is to track a target and react to the target's motion. The reaction is based on some predefined behavior. Ideally, this behavior is modular in that it can be changed easily to involve more complicated behaviors that push the limits of the tracking algorithm. The motivation for research in this area is apparent in everyday life: the realization of an intelligent automobile that follows road lines, a completely automated convoy where trucks automatically follow the back on the truck in front of it, or robots that recognize and manipulate moving parts on an assembly line.

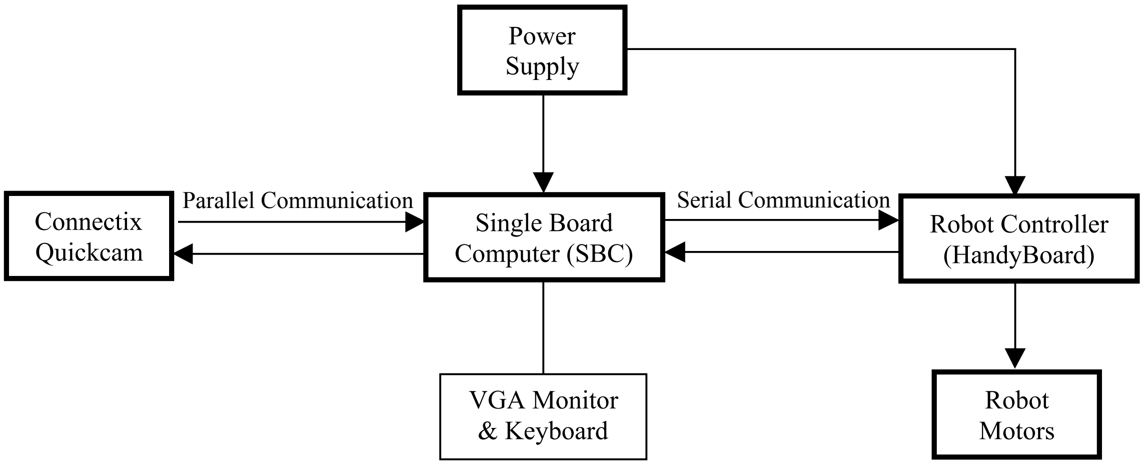
This project delivers a software solution to the tracking and recognition problem in addition to a hardware platform on which it is implemented. The tracking algorithm has been successfully run on a desktop Pentium 233 as well as a 386sx powered mobile robot, and can be scaled up to utilize the processing power capable only of modern servers.

METHODOLOGY

OVERVIEW

An overview of the system architecture is shown in Figure 1. It is a combination of a hardware platform and two distinct algorithms. A tracking algorithm which is responsible for processing image data runs on a single board computer. Responding to a moving target in the image, it sends information to a robot controller which then makes decisions based on this information and finally activates the drive motors. The platform is carried on an autonomous robot constructed from Lego parts. Photographs of the robot can be seen in the appendix.

FIGURE 1: System Component Overview



HARDWARE

The proposed performance criteria specify that the object recognition and tracking arbitrates on color and two dimensional images. To satisfy this requirement, images are captured on a Connectix Color Quickcam. The maximum video frame rate attainable for 80x30 24-bit color is 30 frames per second. The Quickcam is mounted on the robot approximately five inches off the ground and angled slightly downward. This angle yields a maximum viewable range of 12 feet, a minimum range of 18 inches, and a field of vision ~21° wide. Image data is sent over a parallel port operating in bi-directional mode to a single board computer. Handshaking over the parallel port is performed in the tracking software via the Quickcam driver discussed below.

A small processor with reasonable computing power is necessary as the basis for the hardware platform. Size is important due to the desire to construct a mobile robot and carry the processor on board. Processing speed is important because of computationally intensive software. Thousands of comparisons and other operations occur every time a frame is captured. Coupling this with the simple fact that a fast frame rate is pivotal for real time operation and that the hardware platform should be user friendly and easy to debug, a 386sx single board computer (SBC) was chosen. This board is faster and offers more features than all other controllers examined in the same price range. The SBC has several advantages such as a DOS environment for developing and debugging, compatibility with most standard PC peripherals, and the integration of additional serial and parallel devices is extremely simple.

The SBC contains a 80386sx microprocessor (40Mhz), parallel and serial ports, a VGA port and adapter, a 1 Mbyte Flash ROM disk, and a keyboard port. It measures 4 inches x 7 inches. The Flash disk is formatted for and runs X-DOS. The image tracking algorithm is stored on the Flash disk and is executed via the autoexec.bat file or keyboard input. Floppy disk and hard disk controllers are also on board which are used to load the tracking software and for extra storage space. Communication with the Quickcam occurs via the parallel port, which is configured in BIOS to run in bi-directional mode. Communication with the robot control board (HandyBoard) occurs across a serial port configured to: 1200 baud, 8 bits, no parity, 1 stop bit. An error detecting protocol was devised and written to run on top of the standard PC serial port.

The actual robot controller needs to accepts position data from the SBC, make decisions based on the data, and drive motors in response. To this end, the HandyBoard robot controller board was chosen. The HandyBoard contains a Motorola 68HC11 microprocessor, an LCD display, and IO pins to drive motors and read sensors. It receives data via a serial connection to the SBC. The robot control code is initially loaded into static memory on the HandyBoard via the serial link. This program makes decisions based on data input and drives the four attached motors accordingly.

The actual robot is constructed of Lego parts and motors. It is designed to carry the SBC, HandyBoard, and Quickcam. The robot was designed as a treaded vehicle, which allows for greater control than a wheeled vehicle when spinning or driving curves. Also, it is built tall enough for the Quickcam to be mounted several sufficiently above the ground.

A significant design problem exists with delivering power to the devices. The SBC draws power via a standard motherboard connector used in computer power supplies. It draws +5 VDC and +12 VDC at 500 mA. A portable power supply that could fit on the robot to meet this power requirement is unfeasible. Instead, a standard power connector from a PC supply was extended 20 feet and bundled to avoid inhibiting the robot's motion. The Quickcam draws power from the keyboard port on the SBC. The HandyBoard contains a battery pack that can be charged with a +12 V AC/DC adapter. However, the charge is rapidly depleted when driving four motors. To avoid this problem, the AC/DC adapter cable was extended 20 feet and bundled with the SBC power cable. This provides power directly to the HandyBoard.

SOFTWARE

Overview

The software subsystem is mainly comprised of two distinct modules: the tracking software running on the SBC and the robot controller running on the HandyBoard. The two modules communicate via the serial interface between the boards. Error free serial transmissions are ensured through the use of a simple proprietary error detecting protocol. Since the time required to transmit the small amount of serial data per frame is small in comparison to the time required to process each frame, the speed and efficiency of the transfer protocol is insignificant. For each frame, the tracking software on the SBC sends three bytes to the robot controller on the HandyBoard through the error detecting protocol: the X position of the target, the Z position of the target, and a status byte. While the robot is then making a decision based on the tracking data, the tracking code is acquiring and processing a new frame. Although our robot controller is very small and fast, this parallelism allows a much more complex robot controller to be implemented while not adversely affecting overall system performance. An advantage of this modular design is that a different robot controller can be substituted into the system to completely change the robot behavior while utilizing the same tracking code.

Quickcam Driver

A simple frame grabber for DOS was written which communicates to the Quickcam directly over the bi-directional parallel port running in byte mode. The driver configures the Quickcam with a series of parameters read at run time. The driver provides a simple one function C interface for grabbing frames at any resolution.

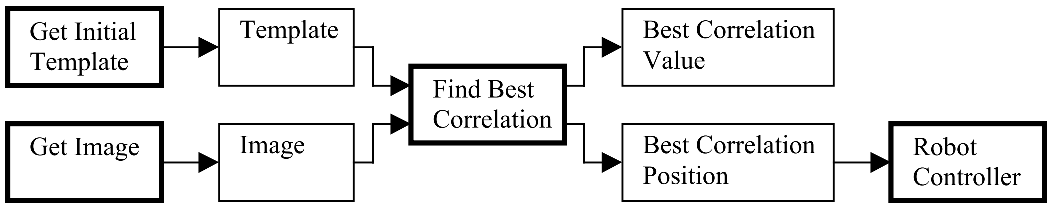
Tracking Code

In selecting a suitable tracking algorithm to be adapted to a visually guided mobile robot, an immediate issue that arises is how to deal with optical flow – the continuous flow of visual data across a sensor. (Ballard & Brown, p. 102) When a target moves across one's field of vision because the target is moving in the physical world, the tracking system contains "local motion." When the body that contains the eye, camera, etc. is physically changing position, the system contains "global motion." Global motion has the effect of adding the perception of local motion to a vision system because background information, while physically remaining in place, will change positions between images. Algorithms exist which can reliably detect motion between frames. These algorithms in one form or another compare the current image to the last image or several images and search for differences. While suitable for vision systems containing local motion, this category of algorithms is not effective for systems containing global motion. In response to this characteristic, an algorithm was developed which ignored all image history concentrating solely on the current captured image.

Since images are to be processed individually and independently of other images, it follows that the algorithm must know what to look for in each frame. Assuming that the algorithm initially knows what to search for, the task of processing optical flow has now been reduced to a problem of pattern or object recognition. Two vastly different methods of pattern recognition found in current work seemed like plausible candidates to be included in a mobile robot tracking system: pattern recognition via neural nets or "dumb" template matching Advantages of neural nets are their accuracy, insensitivity to noise, and adaptability to deformations in the desired object such as rotation, growth (moving closer to the camera), shrinkage (moving farther away), etc. Disadvantages are that neural nets are processor and memory demanding, triple in size and time for RGB color images, and must be retrained for changes in the template, image resolution, etc.. Template matching is an iterative search through an image looking for a template (a sub-image expected to be in a frame) at every possible image location and finding the best fit. Advantages of template matching are that it is computationally simple; no training is necessary allowing on the fly changes in image resolution and template size; the quality of the object recognition can be trivially changed to allow the algorithm to run on faster or slower processors; no floating point operations are required, the speed loss of adding color is minimal, and template matching is easily adaptable for different tracking scenarios. Disadvantages of template matching are that the algorithm is sensitive to noise, lighting, and small changes in object size and orientation. (Ballard & Brown, p.67).

Given the scalability and adaptability of template matching, it was chosen to be the basis of an algorithm suitable for tracking objects on a mobile robot. Basic template matching is depicted in Figure 2.

FIGURE 2 : Basic Template Matching



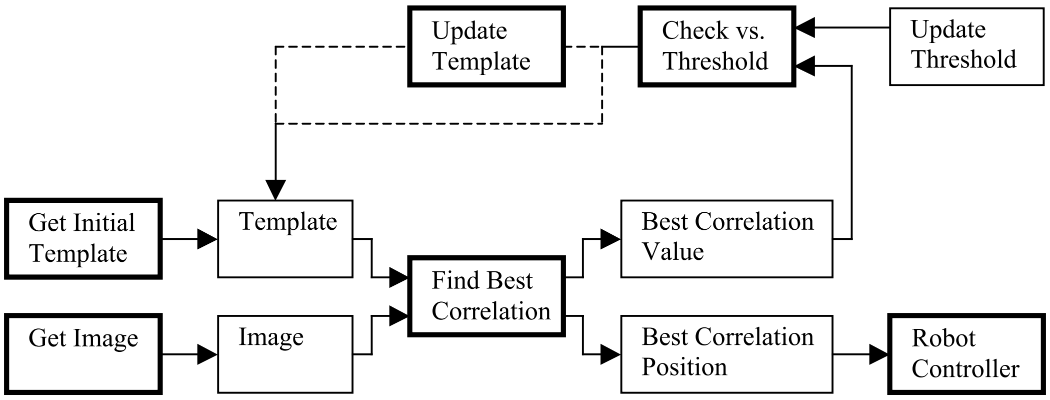
In basic template matching, the initial template is passed to *Find Best Correlation* along with a captured image. *Find Best Correlation* does a simple iterative search through the image generating a correlation value at each possible template location in the frame. The location in the frame of the lowest correlation is assumed to be the object being tracked. The correlation value is defined as the average error per 8-bit RGB pixel. By normalizing the correlation value to error per pixel, it allows the correlation value of different sized templates to be compared. It is important to note that a lower correlation value represents a better match than a higher correlation value. Generating a correlation value at one image location is done by comparing each pixel in the template to the corresponding pixel in the image as follows:

```
Correlation = 0
For each pixel $P_T$ in template
      Find corresponding pixel $P_I$ in image
      Correlation = Correlation
                  + abs($P_I$.red   - $P_T$.red)
                  + abs($P_I$.green - $P_T$.green)
                  + abs($P_I$.blue  - $P_T$.blue)

Next $P_T$
Correlation = Correlation / (Number of Pixels Evaluated)
```

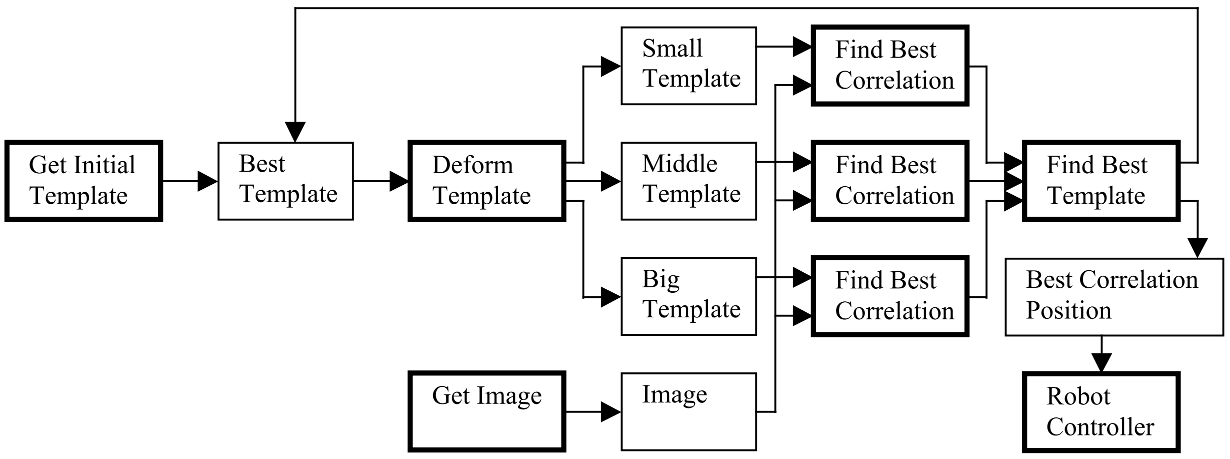
As previously stated, this basic template matching is poor when the object being tracked changes from its initial state slightly. Dynamic template matching (see Figure 3) handles these small changes in lighting, position, orientation, etc. Dynamic template matching is very similar to basic template matching with a small change. When the best correlation position is found, its correlation value is compared against a pre-calculated threshold. If the value is below the threshold, the previous template is discarded and the just-recognized object is made the new template. This threshold value is set such that if the target object changes slightly, it will be recognized and the template will be updated to this new orientation and position of the target. If the correlation value is above the threshold, the last template is carried over to the next frame.

FIGURE 3 : Dynamic Template Matching



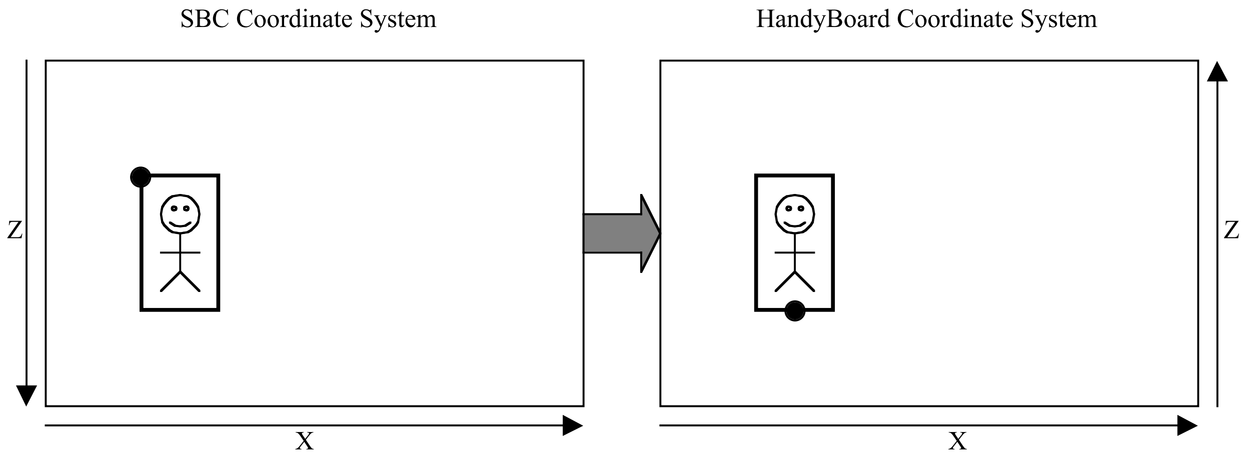
While dynamic template matching provides a huge performance increase with almost no computational overhead, it still has limitations. Problems occur if the target either grows or shrinks causing it to no longer fit squarely in the template. To address this limitation, two new templates are made each iteration: an enlargement and reduction of the main template. The main template is always deformed by a constant proportion and its aspect ratio is always maintained. When a new image is acquired, the lowest correlation values of each of the three templates are found. Whichever of these templates then has the lowest minimum-correlation value becomes the middle sized template of the next iteration. Thus, the templates will grow as the object being searched for approaches and shrink as the object retreats. The tradeoff of the improved tracking by the addition of deformations to the algorithm is a threefold increase in running time. This algorithm is depicted in Figure 4.

FIGURE 4 : Dynamic Template Matching with Deformations



A template is always located in a frame by its upper left hand corner. However, the protocol for communicating between the HandyBoard and SBC requires that the coordinates of the middle-bottom of the template be sent where (X=0, Z=0) is the lower left corner of an image. Thus, a coordinate transformation is required before sending coordinates to the HandyBoard (Figure 5).

FIGURE 5 : Coordinate Systems



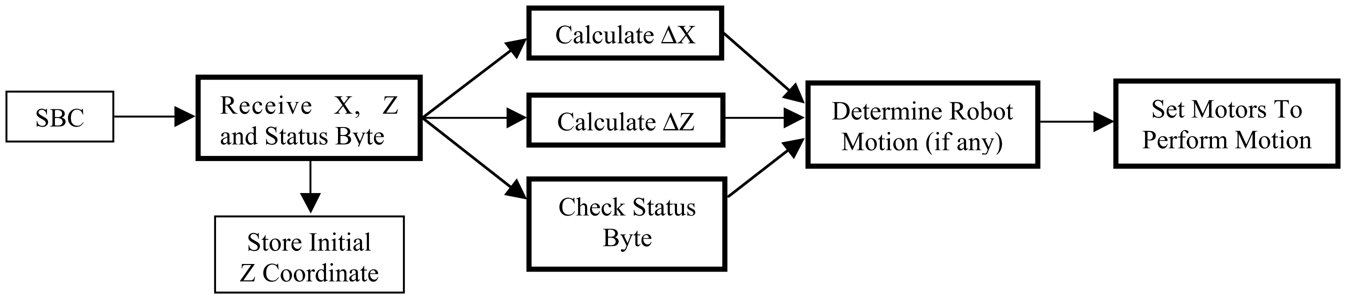
Currently, a monitor and keyboard are required to set the initial template. The user is presented with an image from the Quickcam. Via the keyboard, the user resizes and positions a box over the area of the image which is to become the initial template. Once the initial template is selected, the tracking software executes a loop consisting of grabbing a frame, finding the best location of the best template, and sending coordinates of the object to the Handyboard. The monitor and keyboard are no longer necessary after the initial template is set. One minor yet necessary addition to the algorithm is the *Lost Template Threshold*. If the best template correlation value is above this threshold, it is assumed that the target has left the field of view. When this occurs, the status byte is changed from its default OK value of 0. If the last known location of the target is on the left half of the image, the status byte is set to 1, else it is set to 2. This status byte is always sent to the robot controller along with the target coordinates.

Robot Controller Code

The purpose of the robot control code is to convert the data supplied by the tracking algorithm into a robot response. The expected data for each frame is a coordinate on the X axis (1 byte), a coordinate on the Z axis (1 byte), and a status byte. The combination of all three parameters generates a robot response. The code described here is for a behavior that attempts to maintain the initial distance between the robot and a moving target and to keep the target centered in the field of vision. This behavior was chosen because it is considered to be more difficult than simply driving to a target and showcases the versatility of the system. It is important to remember; however, that this code is modular and any behavior can be modeled.

Figure 6 describes the flow of the robot control program. Data is sent via the serial link every time a new frame is processed by the tracking algorithm. Data is received by the robot controller by polling the serial port with a serial IO function that does not time out. The robot controller echoes every character received as part of the handshaking protocol. The robot controller uses changes in this data to determine how the target is moving. The data for the initial frame is important because it is used to determine the desired range to maintain between the robot and the object. The initial Z position is stored as a constant. Changes in subsequent Z coordinates determine the relative object distance. An assumption is made that z values towards the top of the frame represent objects farther from the robot Z values near the bottom represent near objects. This also assumes that the target is on the same ground level as the robot. In the same way, X coordinates are compared with the X coordinate that represents the middle of the frame to determine where the target is horizontally. The status byte is set by the tracking algorithm based on whether the target has left the robot's field of vision.

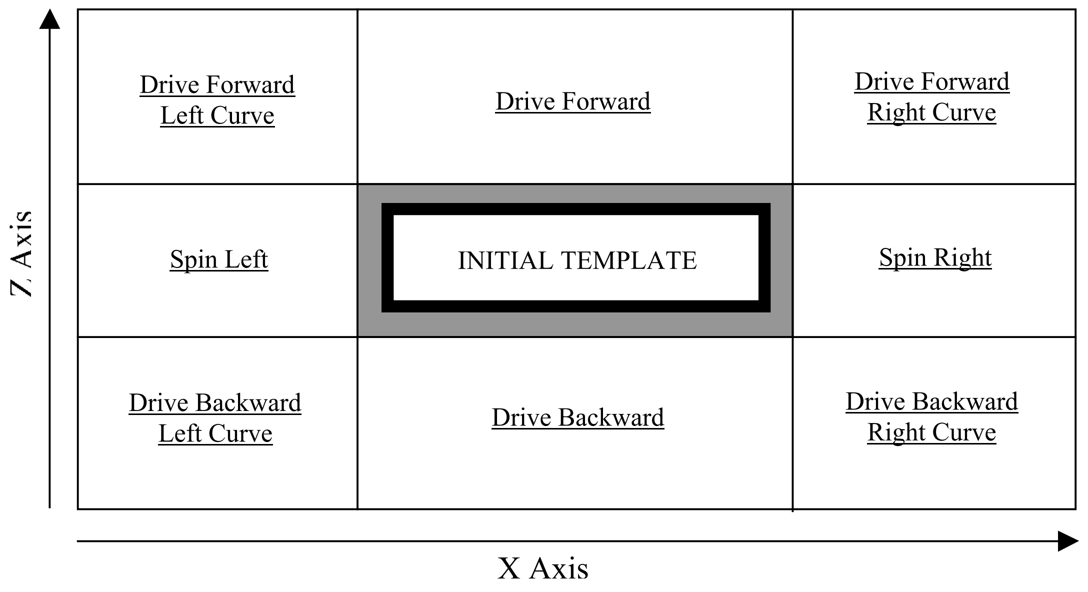
FIGURE 6: Robot Control Code Flow Chart


SBC
Receive X, Z and Status Byte
Store Initial Z Coordinate
Calculate ΔX
Calculate ΔZ
Check Status Byte
Determine Robot Motion (if any)
Set Motors To Perform Motion

The motion of the robot is dependent on thresholds that are defined as constants in the robot code. Thresholds are used to determine when the robot should begin to respond. They also help filter out small drifts in the template that do not represent real target motion. For example, if the Z coordinate increases by two pixels, a robot response would not be appropriate because it is likely that the template temporarily matched a portion of the frame two pixels higher instead of detecting a real motion. The same applies for small changes in the X coordinate. In general this results in a lag between the time the target begins moving and when the robot starts to move. This is desirable and reflects the response time that is involved in most real world situations.

Figure 7 shows the robot response to different screen positions for this particular behavior. The shaded area around the initial template denotes the X and A threshold values. If a change in template position falls within the shaded region, the robot does not move. The remaining areas of the screen show the corresponding movement the robot makes if the template is found in a particular area. The goal is to keep the template in the middle section, which corresponds to maintaining the initial distance and centered in the field of vision.

FIGURE 7 : Robot Motion



There are eight basic movements the robot can make as shown. Each movement is achieved by setting the motor speeds of the four motors to separate levels. For example, a drive forward is achieved by setting all four motors to a constant positive speed. A left spin is achieved by driving the right motors forward and the left motors in reverse. The speeds at which to drive forward, backward and curves are defined as constants in the robot code and need to be calibrated based on the target's maximum speed.

Spinning is a special case in that it is easy to spin too far in one direction and miss the target. To compensate for this, a formula was defined based on the horizontal position to determine how far to spin;

$$\text{Spin Time} = \left(\frac{\Delta \text{x}}{(\text{Frame Width} / 2)} \right) * \text{Max. Spin Time}$$

Spin time refers to how long the motors are turned on for. *Max. Spin Time* is defined as the time that will rotate the robot through the angle that subtends half the width of the frame. The Δx term scales the time proportionally by how far the target is off center.

A status byte is used by the tracking algorithm when the target is no longer in the field of vision of the robot. A value of ‘1’ denotes that the target was last found on the left half of the frame and the robot executes a left spin for the maximum spin time. A value of ‘2’ denotes that the target was last found on the right half and the robot executes a right spin for the maximum spin time. The robot continues to spin until the target is reacquired and the status byte is set to ‘0’.

RESULTS

There are two criteria for judging the success of the implemented tracking system. The first is a subjective opinion if the mobile robot can track and follow a target well. The goal has been met on the basis of three main test categories. The mobile robot can successfully maintain a constant distance when the target drone moves either an oscillatory or random forward-back movement. Second, the robot is capable of aligning itself to a target drone moving laterally. Finally, the robot is capable of following and maintaining a constant distance between itself and a target drone wandering a closed arena. The second criteria for judging the success of the system is the rate at which images can be captured and processed. A higher frame rate allows the robot to react better and follow a quicker moving target.

Running on a 233MHz AMD K6, the tracking code running at 160x60 24-bit color obtains exceptional results. Frame rates of 5 frames per second (fps) are obtained with all tracking features enabled. It is capable of tracking detailed objects such as facial features, soda cans, etc. in a cluttered image. The same tracking and resolution parameters running on the SBC yielded about 0.3 fps which is unacceptable. To compensate for the decrease in processing ability while maintaining reliability of the tracking system, several features and quality parameters were compromised.

On the demonstration robot, it was found that there is a significant tradeoff between the quality of the image tracking and the frame rate. Depending on configurable parameters, frame rates ranged from 1 fps to 5 fps at a template size of 10 x 4 pixels. An acceptable medium between tracking quality and frame rate was found at *MatchResolution* = 2 and *SearchMatchProb* = 0.5 (see Figure 8).

FIGURE 8: Test Results

Match Resolution	SearchMatchProb	Size Deformations	Video	Image Resolution	Correlation Quality	FPS
1	1	Off	Off	80x30	Good	~0.8
2	1	Off	Off	80x30	Good	~2
2	**0.5**	**Off**	**Off**	**80x30**	**Fair**	**~3.5**
4	0.5	Off	Off	80x30	Poor	~4.5

For the purposes of creating a mobile demonstration robot running the tracking software, the algorithm as previously described had to be scaled down greatly to achieve an acceptable frame rate. All frames captured were 80x30 24-bit color images – far below the max 640x480 resolution of the Quickcam. Since the resolution was cut so drastically, the detail of objects being tracked was grossly diminished. The effect this had on image quality was very blurry edges and much greater sensitivity to small changes in lighting conditions and image background. As to be expected, the best tracking performance is achieved when operating the system in a closed, uncluttered environment with ample and consistent lighting. As such, a white-walled arena (roughly 15'x7') was constructed upon a white tile floor. All robot tests and the results in Figure 8 were accomplished in the arena. In addition to controlling the environment, the following features and runtime parameters were modified to improve frame rate: disablement of deformation code, localized template searches, disabling video output, the template correlation procedure, and overexposure.

To further improve frame rate, the deformation code was disabled. By not creating a bigger and smaller version of each best template, the frame rate was effectively tripled. Under the controlled environment, this played little to no effect on the reliability of the tracking; however, this feature must re-enabled in a cluttered environment where the size of the object being tracked is expected to vary greatly.

Localized template searches were temporarily implemented. This consisted of performing a localized search around the last known location of the target. The success of the localized search is directly related to the velocity of the target and the frame rate. Subjective evidence indicated that this shortcut adversely affected the quality of the tracking more than the resulting increase in frame rate helped; thus, it was disabled. However, this is a feature worth considering if the velocity of the target is slow relative to the speed of the mobile robot.

Running with the video output enabled crippled the frame rate; therefore, it was enabled only during debugging runs and turned off for performance tests. Video is still required to select an initial template. On faster hardware, there is no need to disable this feature.

To further increase the frame rate, two degradations are performed to increase the speed of searching for a template in an image. When generating the correlation matrix, the first check for a template at certain image location is done if and only if a random number between 0 and 1 is lower than some probability P (usually 0.25). This has the effect of only evaluating a given template one of every $1/P$ possible locations. This is preferable to simply skipping every $1/P$ template locations because the randomness removes the problem of skipping the same locations every frame. P, referred to as *SearchMatchProb*, and is configurable at runtime. The second degradation, referred to as *MatchResolution* and also configurable at runtime, decreases the number of pixels evaluated at each template check. Setting *MatchResolution* to i causes only the i^{th} pixel of every i^{th} row in the template to be compared to the corresponding pixel in the frame thereby decreasing the total number of pixel comparisons by i^2.

Finally, it was found that overexposing the images tended to remove shadows and noise while increasing the contrast between the drone robot and the background. Setting the exposure time to 30 msec yielded the best tracking results as compared to an exposure time of 13 msec which produces images pleasing to the eye.

CONCLUSION

This project met the goals that were set out in the design goals. A suitable hardware platform has been identified and constructed. The budget for hardware was initially \$250. The total cost for parts came to \$170, the price of the SBC. The Quickcam, HandyBoard, and all Lego parts were borrowed at no charge. Constructing robots from Lego parts was found to be an easy and inexpensive solution to make the platform mobile.

The other major goal was to create a tracking algorithm and robot controller. It was observed that the quality of performance is closely tied to the processing capabilities of the hardware platform. The tracking code is written such that different features can be turned on and off to maximize performance. This ensures the system can be scaled to any platform and still perform successfully. Ultimately, the quality of the tracking is directly related to resolution. The response of the robot controller, however; is more closely related to the frame rate (i.e. a faster frame rate allows the robot to see changes in its field of vision better).

Despite the success of the tracking algorithm in most cases, there are still limitations to what it can track. For example, if the aspect ratio of the target changes, such as a vertical can of Pringles falling over, the algorithm cannot change the shape of the initial template that was chosen. Also, changes in lighting are not compensated for during real-time operation.

The robot controller is ideal in that there is an unlimited number of behaviors that can be modeled and implemented. This allows for switching between applications. An application for the behavior described above might be to automate a convoy of trucks such that a camera on the front on one truck follows a target on the back of another. Another possible behavior is line following which could be useful for operating automobiles or automating mining vehicles. Since the entire system has been designed to scale both upwards and downwards, there exist a multitude of possible applications.

RECOMMENDATIONS

This project demonstrates that hybrids of template matching are a viable solution to vision tracking problems. There are several variations and additions that can be added to the work described here in the future. The exposure time that is preset at runtime could be autocalibrated to the ambient lighting of the environment. Initial template selection could also be automated by implementing a motion detection algorithm in which the robot detects a significant change in its field of vision.

It is important to note that the deformations implemented were only for growth and shrinking of the initial template. For some systems, these deformation may not be sufficient. The same algorithm will work with the addition of other deformations such as one which handles sharp changes in lighting. One interesting deformation would be to be able to handle object whose rotations greatly change their aspect ratio. Deformations that affect template aspect ratio such as stretching or rotating could be employed.

In terms if the hardware platform, it is desirable to power the robot on board so that it is truly autonomous. A behavior that was not modeled on the robot controller involves varying the driving speed based on how quickly a target is moving. This was not feasible due to the relatively slow frame rate, but would be a good behavior to implement.

Of course the single most important modification to increase performance is to increase the processor speed. This would allow higher resolution images to be captured at much better frame rates. An exciting possible addition is an auto-zoom feature. When a target moves farther away, the camera could capture images at a higher resolution and crop them, thereby creating a magnification of the previous template.

Ultimately, we have demonstrated contrary to the opinions of many current vision researchers, template matching is not an obsolete technique. It is a powerful, versatile, scalable tool, and possibly most important of all, easily learned and implemented by researchers new to the field of visual motion and tracking.

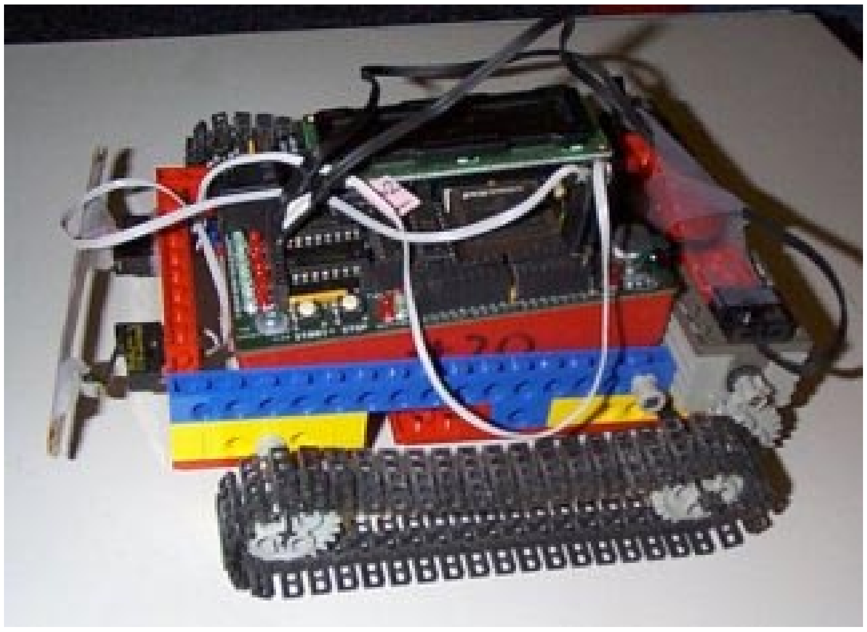
APPENDIX

Additional information about the HandyBoard can be found at:
http://el.www.media.mit.edu/groups/el/Projects/handy-board/

Additional information about the Connectix Quickcam:
http://www.logitech.com/

References:
Aleksander, I. (ed) Artificial vision for robots. Chapman & Hall: New York, NY, 1984.
Ballard, D.H., Brown, C.M. Computer vision. Prentice-Hall: Englewood Cliffs, NJ, 1982.

Photograph of Drone Robot:



Photographs of Tracker Robot: