TJPRO-AVR: A Wireless Enabled Mobile Robot for Vision Research

Scott Jantz and Keith L Doty

Mekatronix Inc.
URL: www.mekatronix.com
Tel. 352-376-7373
&
Machine Intelligence Laboratory
Department of Electrical and Computer Engineering
University of Florida, USA
Email: scott@mekatronix.com , doty@mekatronix.com

2006 Florida Conference on Recent Advances in Robotics
May 25-26, 2006, Florida International University

Abstract

In this paper we describe a vision system based on combining a mobile autonomous robot with wireless links to a host PC running MATLAB™. The TJPro-Avr™ consists of a mobile robot platform with 2 motors, a battery pack, IR sensors and an Atmel 8-bit AVR microcontroller, the ATmega128, for control. We have expanded the basic TJ Pro-Avr™ for these experiments by adding a wireless serial link and a wireless camera. With this platform we have performed several basic robot behaviors.

1. Object Avoidance using structured light.
2. Object detection and avoidance using motion and object detection.
3. Object tracking and interception using color differentiation.

Our objective in using these simple vision algorithms is to demonstrate how our mobile robot platform, in conjunction with MATLAB, permits mathematically sophisticated complex processing of real-time mobile robot sensor data with a user-friendly interface at an affordable cost. Our choosing of a color detection algorithm as one of the experiments allows us to compare our current platform with an on-board PDA doing the vision processing. That approach was detailed in our 2002 paper: "PDA Based Real-Time Vision for a Small Autonomous Mobile Robot"[10]

Introduction

The TJPro-Avr™ mobile robot, configured with wireless data and camera image communication to a host personal computer executing MATLAB, provides an excellent platform for exploring visual behaviors in autonomous mobile robots. The wireless camera and Bluetooth serial link allow the robot to be connected to any computer within several hundred feet. This remote connection allows the robot

to take advantage of the processing power on a PC without carrying around the PC's weight or power supply. This allows the robot to be small, inexpensive and harmless, while still very capable.

Using MATLAB to control the robot allows very fast development of vision algorithms. Since many common matrix and vision algorithms are built into MATLAB we can focus on experimentation and not just writing code. Not only does MATLAB allow easier and faster vision algorithm development, but its C-like syntax allows robot control to be easily implemented as well.

TJ Pro-Avr™ Robot Platform

The TJ Pro-Avr™ / Wireless platform (**Figure 1**) uses the Mekatronix Mekavr128 microcontroller board, based on the Atmel Atmega128 microcomputer, as its main controller. Mekatronix's Robot ASCII Command Interpreter (RASCII) executing on the robot controller during the experiments allows full remote control of the robot and complete, remote access to its sensor values through the wireless serial data port. Through RASCII serial commands we can control the motor speed and query all the sensors. To extend this control to the computer running MATLAB the robot sends the serial data through the Xbee Bluetooth module. On the computer side another Xbee Module converts the data to a USB serial port. The use of Bluetooth allows the robot communication to run at a full 115200 baud. This high-speed communication enables real-time control of the robot with little or no latency to motor commands.

To transmit video we use a Grand Tec RFC-3000 wireless camera. The receiver for this camera includes a USB video capture device to digitize the images. MATLAB includes a video capture library, which allows us to easily capture images from a variety of sources with just a few lines of code.

Figure 1: TJ Pro-Avr™ robot vision system.

.

Experiment 1: Structured Light

To show the power of the Wireless TJPro-Avr™ platform we will start with a very simple example. We will show how to program the robot to avoid obstacles using IR light. During this experiment the only addition to the robot was an IR (940nm) filter placed in front of the camera lens. The TJPRO includes IR LEDs and detectors for collision avoidance. Normally the TJPRO uses 40Khz, 940nm light to avoid obstacles.

Receivers tuned to 40Khz detect the illumination of nearby objects and the output voltage of the sensors roughly measures the distance.

Using the camera system we ignore these receivers and simply use the LEDs to illuminate the surroundings in a frequency of light that will pass through the filter. This filter allows the vision algorithm to be very simple. When there are no objects nearby to reflect the IR light the image appears completely black. Only when an object is close by does the robot see any reflection (**Figure 2**).

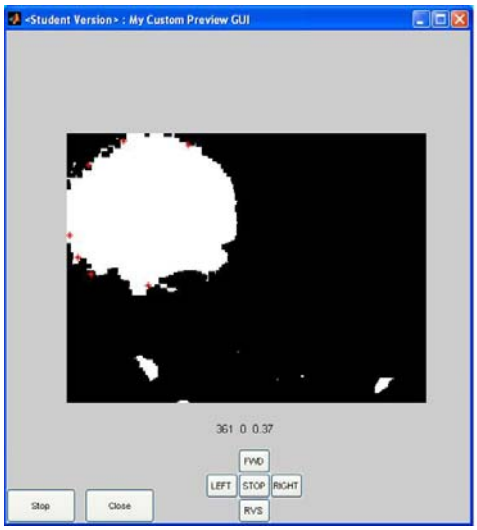

Figure 2: screen shot showing an IR illuminated object to the left of the robot.

In Figure 2 the main blob consists of an IR illuminated wall. The circle results from the circular radiation pattern of light cast by the IR LEDs on the front of the robot.

To get to this processed image from the original captured image in (**Figure 3**) we use several image processing techniques.

Figure 3: Original image showing IR illumination through filter.

After capturing the image from the camera the MATLAB program converts the RGB image to a grayscale image:

```
vid                                    =
videoinput('winvideo',1,'RGB24_320x240');
frame = getsnapshot(vid);
BW = rgb2gray(frame);
```

Then the image is run through a simple black and white threshold detection:

```
level = 0.37;
BW2 = im2bw(BW,level);
```

We can use this simple technique because we are using structured light, which greatly simplifies the image. Then our program applies several morphological operators:

```
I2 = imclose(BW2,se);
final = imabsdiff(I2, OldI);
I3 = imfill(final,'holes');
I4 = bwareaopen(I3,50);
```

With a differential to the last frame in-between the close and fill operators. Morphology allows us to simplify the image and eliminate much of the noise. Morphological operators treat binary images as sets which are members of a two dimensional space. [9]. For example we can use the *erosion* operator. Erosion is defined as

$$\{x \mid (B)_x \subseteq A\}$$

Where erosion results in a set of points for which *B* fits entirely inside *A*. Erosion eliminates any pixels not part of a continuous object and most of the "shot" noise produced by the camera. MATLAB has more advanced operators at our disposal such as close and fill, which combine an erosion operator with a dilation operator.

To obtain simple data from this black and white image MATLAB can calculate the centroid and area of each blob detected by our morphology. When MATLAB executes:

```
L = bwlabel(I4);
graindata = regionprops(L,'basic');
```

We are left with an array of centroids and areas for all the blobs detected. This data is used to determine how the robot should react. The movement algorithm is very simple; the robot flees from any large blobs.

In addition to demonstrating some important image processing techniques this experiment allows us to test the performance of the robot. Because the end result is simple IR collision avoidance we can directly compare the speed of the robots reactions to those using the IR sensors directly, without image processing. Under the control of MATLAB through a wireless interface the robot moved around objects with about the same dexterity as the on-board computer control with IR sensors. This result suggests to us that our mobile robot vision system can be used for more complex and useful vision algorithms.

Experiment 2: Simple Visual Flow

Our second experiment involves implementing collision avoidance based on animal-like behaviors using natural light. We will combine several basic vision ideas to produce simplified data from a continuous video feed. The MATLAB program will capture a frame, as before, and then perform two new operators:

```
I1 = edge(BW2,'canny',0.3);
I2 = imclose(I1,se);
```

The first line performs the Canny edge detection algorithm with a threshold of 0.3 this is a high threshold which eliminates almost all the false edges (ones due to texture rather than actual edges). `Imclose` is another combined morphological operator that helps rid the frame of false edges. Then:

```
final = imabsdiff(I2, OldI);
finalerd = imerode(final,se2);
I3 = imfill(final,'holes');
```

The program performs these operators. `Imabsdiff` simply differentiates the images (new frame to old frame) in a similar way to the retina therefore if there is no movement the entire image will be black. After that we do a standard erode operator that eliminates some noise. In the final line we do a simple fill of all closed areas to amplify our moving edges. A moving edge will likely carve out a polygon that can be filled (**Figure 4**).

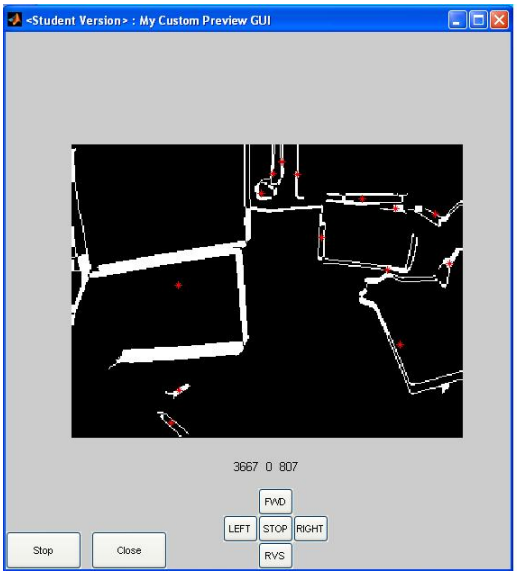

Figure 4: Edge detection plus differential

This simple shortcut allows us to take complex edge data and distill the motion data contained in into area data. The larger (and closer) the edge the larger the area carved out by the moving edges. Also, the more the edge moves from last to current frame the closer the edge is to the robot (due to parallax). Area and position of filled areas, then, yield important data on the edges: how fast they are moving and their size. To get to area and centroid we use the same code as the last experiment. After obtaining our arrays of centroids and areas for all the blobs in the final frame we sort them by ignoring the top half of the frame (mostly distant objects) and calculate 3 weighted vector sums for the remaining ones. These correspond to the amount of clutter in the left, center and right visual field of the robot. The robot control algorithm uses these numbers to choose an action that will reduce the likelihood of a collision.

Experiment 3: Robot Soccer

In this experiment we combine several of the techniques used in the other experiments with color space remapping. The goal is for the robot to be able to recognize a red ball and take it to a blue goal. For our robot to see a colored ball, it first must differentiate between the color of the ball and all other colors.

RGB values produced by the camera cannot be easily interpreted as to the redness, greenness or blueness humans see. To enable the robot to differentiate between object colors roughly according to human vision, the RGB values must be remapped to another color space. Our two best options are YUV (also know as YIQ) and HSV (also known as HIS) color spaces [1] [2] [3] [6] [12] [9].

In HSV the values of Red Green and Blue are transformed into Hue, Saturation and Value where, theoretically, Hue completely represents the color independent (at least in theory) of lighting. Again MATLAB helps us quickly implement this. After capturing the frames just like in the other programs we convert to HSV color space through the following function:

```
frame2 = rgb2hsv(getsnapshot(vid));
```

And to determine what hues correspond to various objects we can use the following tool:

```
imtool(frame2);
```

to produce the image tool display in (**Figure 5**)

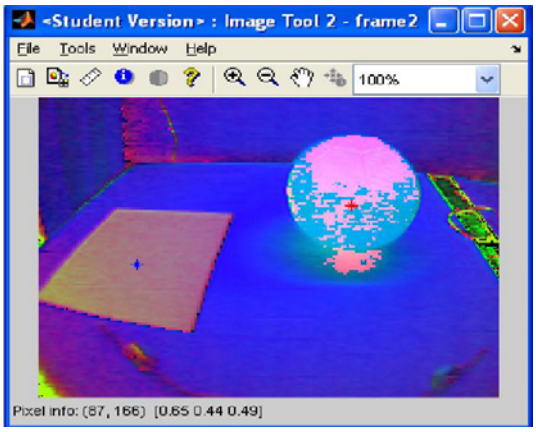

Figure 5: Display from `imtool` of an HSV converted image.

The pixel info at the bottom allows us to see values of any area we point to in the image. This is a very powerful tool for debugging and setting color ranges. We then convert this image to 2 black and white frames both containing only red or blue pixels (**Figure 6**)(**Figure 7**)

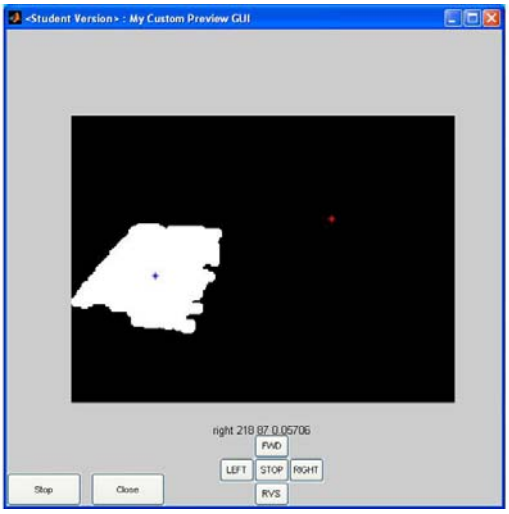

Figure 6: Blue pixel image.

Figure 7: Red pixel image.

After data filtering, similar to the techniques used in the other experiments, the processed images are clean enough for the blob detection. The program computes the areas and centroids for all the objects and searches through them for the largest blue and largest red objects. These points are shown by red and blue stars in the output to help us debug.

Now that the entire image has been distilled to two sets of coordinates we simply program the robot to move towards the red ball, centering itself in front of the ball and moving forward until robot and ball touch. Once the ball is touching the robot the algorithm starts searching for the blue area (the goal) and moves toward it. When both the red ball and a portion of the blue square are low enough in the frame (indicating they are close) the robot declares victory and backs up, leaving the ball on or next to the goal.

User Interface

In addition to the powerful image acquisition and processing tools MATLAB also allows the easy

development of a user-friendly interface. In many of the previously illustrated screen shots you may have noticed several buttons on the screen. The Stop and Close allow us to shut down the program running the robot, while properly closing open device drivers (serial port and video capture). This allows the program to gracefully exit and re-run without memory leaks or open device drivers that may cause errors. These buttons are easily created using the MATLAB GUI development and these buttons simply call a *callback* function to execute an action:

```
uicontrol('String', 'Close',...
    'Callback', 'kill()',...
    'Units','normalized',...
    'Position',[.17 0 .15 .07]);
```

This is similar to most GUIs such as Windows or Linux widgets. The other buttons allow remote manual control of the robot while looking at the output of the vision algorithms executing on the host computer. This is an extremely powerful mobile robot vision algorithm development and debugging tool as it allows the developer to get "inside the head" of the robot and see through the robot's eyes by observing the visual effects of various algorithmic strategies.

Comparison to onboard PDA

We have implemented a similar algorithm to the robot soccer experiment on a TJPro™ in a previous paper [10] . That platform consisted of a 68HC11 based TJPro™ robot and a Handspring PDA. The HandSpring PDA and *Eyemodule* camera made a good combination for simple vision work. The size of the PDA and camera were perfect for our TJ Pro™ robots. The GNU and PalmOS development environment is not as sophisticated as Visual C++ for windows, but it is certainly much better than the development tools for a custom microcontroller board. The PalmOS cannot easily deal with memory blocks greater than 32K in either a static mode or in dynamic memory allocation. The HandSpring speed may easily be increased to 54Mhz by using an overclocking program. Some experimentation would be needed to find a working baud rate at this speed since the serial clock would also be skewed. The cost of the system was a main advantage being less than half the cost of a low level laptop or desktop computer.

Here is how the the HandSpring version and the current configuration compare:

TJPro™ PDA advantages:

1. Power efficient computing.
2. External computer not needed.

TJPro™ PDA disadvantages:

1. Slow processor (54Mhz vs 3Ghz PC)
2. Memory constraints (32K objects)
3. Only C++ programming no vision libraries.
4. Vision algorithm development complex and difficult.

Wireless TJPro-Avr™ and MATLAB advantages:

1. High processing speed coupled with PC technology (3GHz currently)
2. Large memory coupled with PC technology (GByte or more)

3. Ease of software development: image acquisition and analysis libraries take much of the drudgery out of programming and allow the developer to go straight into sophisticated algorithms.
4. Portable: Need more processing power, buy a new computer there are little or no changes needed to the robot or MATLAB M files in order to move to a new computer or even a different OS.

Ties:

1. Size: The size of both platforms comparable.
2. Cost: About equal cost for both platforms, even after obtaining a MATLAB license, assuming the user already has a PC.

Future Work

The work described in this paper only scratches the surface of what can be done with the wireless data and camera enhanced TJPro-Avr mobile robot platform working in conjunction with a host PC running MATLAB. This system clearly opens up the possibility of affordable, advanced image processing applications, such as stereovision. Our *M-file* code should have no problem supporting two cameras.

All new computers support *hyper* threading and many new computers are coming with 2 and, soon, 4 physical processors. To improve real-time performance, the vision algorithms can run multiple threads of MATLAB on multiple physical processors. Bring able to take advantage of all these processors on the newest platforms would propel this platform ahead of any stand-alone robot platform at a fraction of the cost.

References

[1] Bandlow T. et al, *Fast Image Segmentation, Object Recognition and Localisation in a RoboCup Scenari*o, Lecture Notes in Artificial Intelligence Volume 1856, Springer, Berlin, 2000.

[2] Bartelds, R. et al, *Clockwork Orange: The Dutch RoboSoccer Tea*m, RoboCup 2001, Springer-Verlag, Berlin, 2001.

[3] Berry A., *Soccer Robots with Local Visio*n, Honours [sic] Dissertation, Univ. of Western Australia, Dept. of Electrical and Electronic Engineering, 1999.

[4] Carlson, N. R. *Physiology of Behavior* (5th ed.). Needham Heights, MA: Allyn and Bacon, 1994.

[5] Fairhurst, M. C. *Computer Vision for Robotic Systems: an Introduction.* New York: Prentice Hall, 1988.

[6]Foley, J. D., van Dam, A., Feiner S. K., Hughes, J. F. *Computer Graphics Principles and Practice*. New York: Addison-Wesley, 1997.

[7] Foster, L. *Palm OS Programming Bible*. Foster City, CA: IDG books, 2000.

[8] Fu, K. S., Gonzalez, R. C., & Lee, C. S. G. *Robotics: Control, Sensing, Vision and Intelligence*. New York: McGraw Hill, 1987.

[9] Gonzalez R. and Woods R., *Digital Image Processing*, Addison-Wesley, Reading
Massachusetts, 1992.

[10] Jantz, S. and Doty, K. L. *PDA Based Real-Time Vision for a Small Autonomous Mobile Robot*, FCRAR 2002.

[11] Mead, C. *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1989..

[12] Pratt W., *Digital Image Processing, Second Edition*, John Wiley & Sons, New York,
1991.

[13] Reiser, M.. *PC-eye Bot / Toro*. [On-line]. Available: http://mil.ufl.edu/imdl/papers/IMDL_Report_Fall_99/Reiser_Mike/toro.pdf, 1999.