Smart Camera for Trax Playing Robot

Demonstration Paper

Donald G Bailey
School of Engineering and Advanced Technology
Massey University
Palmerston North, New Zealand
D.G.Bailey@massey.ac.nz

***Abstract*—Smart cameras are cameras which not only perform the sensing, but also integrate much of the image analysis and control. This paper describes the replacement of a conventional vision system within a Trax playing robot with an FPGA based smart camera. Integrated within the camera are the following functions: camera calibration, tile detection and localization, human player detection, robot control, and game play. The result is that the smart camera not only captures the images, but also analyses the image data and performs all of the functionality required to run the robot completely autonomously.**

Keywords—camera calibration; autonomous operation; image analysis; robot control

I. Introduction

Trax is an abstract two-player strategy game. It is played with a set of identical square tiles, with curved paths on one side, and straight paths on the other side [1, 2]. The objective of Trax is for each player to form a closed loop or line of their colour, while preventing the opponent from completing the objective in the opposing colour.

Trax has relatively simple rules [1, 2]. In each turn, a player plays a primary tile adjacent to the tiles already in play, to extend the path. As a result of the primary tile, one or more forced tiles may also be required. These occur whenever two same-coloured paths enter an empty space; there is only one way to link such paths, hence the tile is forced. Forced tiles are considered as part of the same turn as the primary tile. Trax is a perfect information (there is no hidden information) pure strategy game (there is no luck involved). While the rules are relatively simple, Trax has considerable strategic depth, making it an interesting game for analysis and robot play.

The physical robot was built some time ago [3, 4], with the goal of playing autonomously against a human opponent with real tiles. However, this was based on a conventional computing platform, for which the associated computer was relatively bulky. The goal of this demonstration was to integrate all of the vision and control within an FPGA based smart camera to considerably reduce the robot size by eliminating the need for a PC, and improve its portability.

II. Demonstration System Setup

The camera is based around a Terasic D5M Bayer colour camera module [5]. This captures a 5 megapixel image of the playing area, and is the primary input for the robot. The image is processed by a DE0 Nano FPGA board [6], which is based on a Cyclone IV FPGA. All of the image analysis operations are performed on the DE0 Nano.

The robot is a gantry style robot, with four degrees of freedom. The primary movement is governed by two stepper motors for the X and Y directions. Having a gantry robot simplifies the kinematics, and enables movement with an equal precision of approximately 0.2 mm anywhere within the 700 mm × 500 mm playing area. There is a small (20 mm) vertical movement provided by a Z stepper motor to lift the tiles when positioning the tile being played. Finally, the orientation of the placed tiles is controlled to a precision of 0.8° by a θ stepper motor. A small vacuum gripper is used to pick up the tiles.

During game play, the robot determines whether or not it is the robot to play or the human to play. The position (and orientation) of each tile is determined through image analysis. The detected tiles are divided into two categories (see Fig. 1): the group of contiguous tiles in the centre of the playing form the current position, and the remaining tiles scattered around the playing area are available for play. Any changes to the current position are used to detect whose turn it is to play.

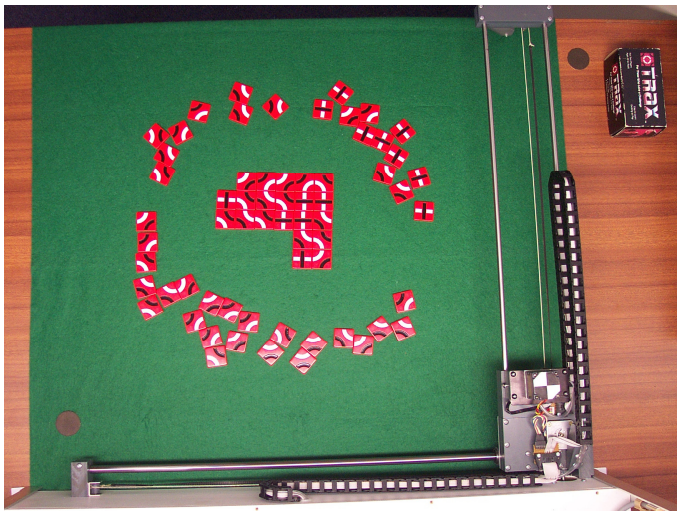

Fig. 1. Top view of the robot and Trax playing area.

If it is the human player's turn, the robot waits until a move has been completed and then updates its internal state of the game. If it is the robot's turn, the most recent move played by the

978-1-4673-9091-0/15/$31.00 ©2015 IEEE

human player is passed to the game engine, which then returns the move to be played by the robot. For each tile to be played (the primary tile, and any forced tiles) a tile is selected from those available, and the robot arm moved to pick the tile up and place it in the appropriate place in the centre of the playing area.

III. Image Analysis Architecture

The image analysis functions are shown in Fig. 2. The processing is split into four parallel branches: camera control; robot manipulator location; tile location, and human player detection. The role and implementation of each of these is described in the following section.

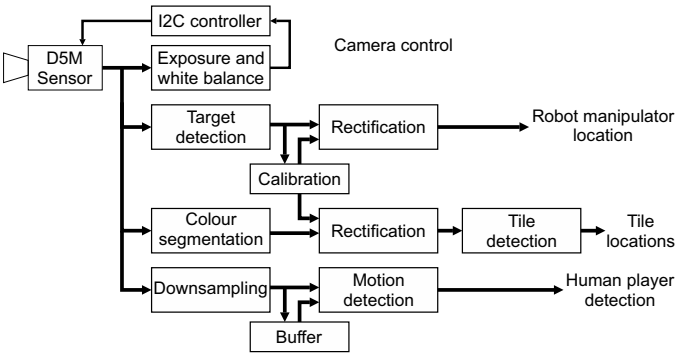

Fig. 2. Image analysis functions within the smart camera.

A. Camera control

The camera control module configures the camera to capture 15 frames per second at 2592×1944 resolution (using a 96 MHz clock). The pixel stream is of raw Bayer patterned pixels with 12 bits per pixel.

The exposure and white balance is controlled using a white patch model [7]. This makes the assumption that the brightest region within the image is white (which will be from the white paths on the tiles), and performs white balancing by adjusting the gains of each of the RGB components to make the brightest point white.

Rather than simply looking for the brightest pixel in each channel some allowance is made for specular reflections and other highlights that may be present in the image. Therefore, each of the raw RGB channels in the pixel stream from the sensor are analysed by counting the number of pixels above the threshold levels 0xFC0 and 0xE00. If the pixel count above the 0xFC0 threshold exceeds 1023 then that channel is over-exposed and the gain for that channel is reduced for the next frame. Conversely, if the pixel count above the 0xE00 threshold is fewer than 1023 then that channel is under-exposed, and the gain is increased for the following frame. The dead-band between the two thresholds allows the gains for each channel to stabilise.

B. Robot manipulator location

To determine the correspondence between the physical position of the tiles and robot, and those within the image, the camera system must be calibrated. This is facilitated by mounting a target on the robot manipulator (visible in the bottom right corner in Fig. 1). The target is detected in the image using a 33×33 matched filter, as shown in Fig. 3. This filter is separable, enabling an efficient implementation. Horizontally, a 16 pixel sum is calculated recursively, and a delayed version of the sum is subtracted.

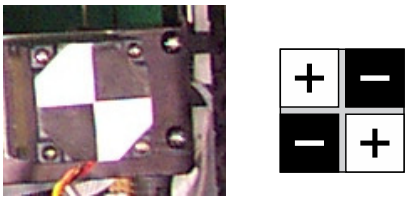

Fig. 3. Calibration target on the robot manipulator, and the corresponding matched filter used for detecting its location.

The vertical filter is the same (apart from replacing the delays with row buffers). However, since the approximate horizontal position of the robot manipulator is known, the memory for the vertical filter can be reduced by a factor of approximately 10 by only using 256 pixel wide row buffers. This effectively limits the operation of the filter to a vertical strip region of interest within the image.

For calibration, the robot manipulator is moved in a 3×3 grid covering the complete playing area. This provides sufficient information for determining both the lens distortion parameters and determining the perspective distortion [3, 4]. The calibration parameters are solved using a small custom soft-core processor.

After calibration the calibration parameters are used to rectify the detected location to give the robot position in table coordinates. Calibration data is also used to rectify the image for tile location.

C. Tile location

Detecting the tiles is made easier by the tiles having strongly contrasting colours and performing white balancing within the camera. The incoming pixel stream is first demosaiced from a raw image to a full colour using edge directed bilinear interpolation. Pixels are then classified as belonging to either red, white, or black using simple thresholding within RGB colour space. Any pixels not within the red, white or black thresholds are classified as belonging to the background. Colour classification is performed at this stage to reduce the volume of data down to 2 bits per pixel.

Next, the image is rectified using a forward mapping. The location of each camera pixel is mapped to table coordinates, and written to a 1024×64 memory buffer by rounding to the nearest location. This window corresponds to a rolling window slightly larger than one tile, which is all that is required for tile detection. Note that because of distortion, the path within the memory buffer will be curved in general.

One common problem with using the forward mapping is holes in the output which have no input pixels map to them. This is overcome by ensuring that the magnification is less than one, such that the buffer resolution is lower than the incoming image. There is still sufficient resolution that when multiple input pixels map to the same output pixel, the last pixel overwrites any previous values.

Candidate tile locations are detected using a set of 8 simple 4-point matched filter templates which must have two pixels of each coloured path. If the pixel is written to the memory buffer

is a path pixel, then these templates are immediately tested to indicate candidate tile centres.

The centre of gravity of sets of contiguous candidate tile locations are defined as the tile centre. Once a tile is detected, the samples within a circular mask centred on the tile centre are used to determine the tile orientation more precisely. For this, the second moments of the coloured paths are calculated using a set of four linear filters in parallel to weight the path pixels, with the results of the filters combined to give the orientation.

As they are found, the detected tile locations and orientations are passed to the tile processor which determines which belong to the current game position, and which tiles available are for play.

D. Human player detection

An absolute difference between frames is used to determine the presence of the human player within the playing area.

A frame difference requires buffering a complete image, which cannot be accommodated on-chip on the FPGA. The frame buffer is located in off-chip DRAM. Change detection does not require the full 5 megapixel resolution, so the raw image is down-sampled by a factor of 4 (using a 4×4 average) before being saved in off-chip memory. At the same time, the previously saved frame is loaded and the absolute difference is calculated. The absolute difference is accumulated over the whole frame, and compared with a threshold to determine if there is any human activity.

To reduce the row buffer requirements, the 4×4 average uses the same window as the Bayer demosaicing filter.

IV. Game Play

The game play is split into four main modules: game representation, game engine, motor control, and game state machine. The relationship between these and the image analysis are shown in Fig. 4, and expanded on in the following section.

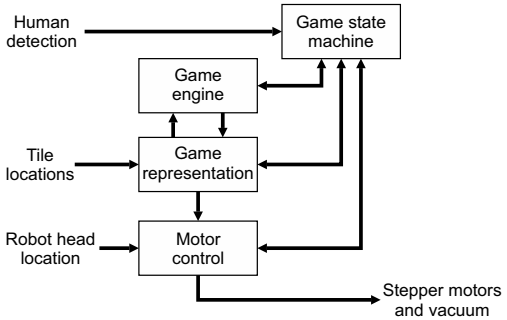

Fig. 4. Modules associated with game play and robot control.

A. Game representation

Within the Trax playing robot, the game is represented simply as a matrix of tiles. Although the game can potentially be infinite, the physical limitations of the robot mean that the position cannot get larger than about 18×18. Therefore the game is represented within a single on-chip memory block by a 32×32 array of bit-masks (with 8 bits per tile – one for each colour in each direction). This enables forced tiles and illegal moves to be detected easily. The origin corresponds to the first tile played, with the tiles indexed using two's complement arithmetic. To maintain logical coordinates for the game engine, the topmost and leftmost rows are recorded in registers, and are subtracted from the absolute coordinates to get the relative coordinates used by Trax notation.

A mapping is required from table coordinates and tile coordinates. If there is no movement from the human and it is the human's turn to play, then the list of tiles from image analysis is processed. To analyse a position, first the detected tile list is sorted in order of distance from the centre of the playing area. The tiles are then mapped into an empty tile array, building up the position starting from the centre of the playing area. Throughout this process, the mapping between table and tile coordinates is determined. If a tile is not adjacent to a tile already in play, that tile is added to the available tile list.

Once all of the tiles have been processed, the position is checked to verify that all forced tiles have been played. If so, then the current position is matched with the existing position. This allows for all of the tiles to be shifted to re-centre the tiles if necessary. Finally, it is checked that the current position has changed, and that the position can legally be derived from the previous position.

If the position has not changed, then the robot waits for the human player to make their move. If there are any errors within this process (incomplete position, or invalid move) the player is notified. Once a new valid position is determined then the human's move is considered complete. The current position replaces the existing position, and the move played is sent to the game engine for the robot's move.

B. Game engine

The role of the game engine is to analyse a position and determine a strong move to make. This requires more complex data structures than the simple tile array as described above.

Commonly min-max or alpha-beta searching is used to analyse a position to determine a good move [8]. Unfortunately, such techniques do not work particularly well with Trax, especially against stronger players [3]. This is because some threats consist of sequences of up to 20 moves to complete [2], requiring a search of that depth to recognise the threat if conventional searching was used. With a branching factor of 60 to 80 for each move (in the middle of the game) such a search is intractable, even when accelerated on an FPGA. Truncating the search early leads to the horizon effect, where losing moves can appear to be reasonable.

Most Trax threats consist of patterns of tiles [1]. Therefore, rather than use conventional tree-based search methods, pattern matching is used to immediately recognise complex threats without any look-ahead. Although humans generally consider patterns in terms of the tiles, path equivalence means that it is not the actual path that is important, but where the paths emerge on the edges of the playing area [9]. This allows the edge of the playing area to be scanned for patterns associated with the different threats. Equivalence principles [9] also enable many related sub-patterns to be represented by a base pattern with variations, enabling many different variants of threats to be recognised efficiently.

To facilitate the pattern search around the edges of the playing area, the representation used by the game engine consists of the tile array, with the edges marked with the corresponding location of the other end of the path. A database of threat patterns is encoded within a finite state machine which is used to match the patterns. The edge information, combined with the path ends, is matched with the database patterns using the finite state machine to identify threats. A search is started from each of the spaces around the playing area. When a threat pattern is detected, it is added to a list of potential threats, along with the move next move required to use the threat (provided as an output from the search finite state machine). This enables a complex threat to be recognised as a sequence of moves, with each stage of the threat being recognised as a threat in its own right.

Once all of the edges have been examined, the list of potential threats is evaluated and combined with a simple evaluation function of each path (used to give weight to growing lines and number of corners) to provide a score for the current position.

To determine a suitable move to play, one level of look-ahead is applied to the current position. That is every possible move is considered, and is evaluated using the search finite state machine, and the move with the highest score is selected. (To make the game play more interesting, a small random number is added to the score for each position before selecting the highest. This usually results in one of several good moves being selected. If there is only one good move available, the random number is small enough to prevent selection of a bad move.) The result is a strong tactical player. Longer term strategic play may be enhanced by using this more complex evaluation function within a tree based search.

The game engine is implemented separately from the rest of the robot smart camera. Since designing a Trax game engine is part of this year's design challenge at the International Conference on Field Programmable Technology (FPT), enabling the competitors to plug their game engines into the Trax playing robot would make the design challenge more interesting. Communication between the Trax robot and the game engine was implemented using RS232 communication, according to the specifications for the design competition.

C. Motor control

Once the game engine returns the move to play, the robot's game representation is updated, a list of the tiles to be played by robot is determined.

For each tile to be played, the nearest tile of the appropriate type (curve or straight) from the available tile list is determined. The stepper motors are controlled to move the robot manipulator to the available tile, pick it up, and move it to the appropriate position within the playing area.

The stepper motors are driven open-loop. However, the motors are low powered for safety (so that they do not harm the human if a collision occurs), and this can occasionally result in missed steps. High-level visual feedback is used to verify that the motor has moved correctly, using the position of the robot manipulator from the image analysis. Any discrepancy between the expected and detected position triggers a motor reset process before continuing. Motor reset steps the motors from the current robot position to the origin, and resets the open-loop step-count.

The stepper motor driver takes the difference between the current step-count and the target step-count for each motor to determine how many steps are required for the motion. The step rate starts slow, accelerates during the early part of the motion, and then decelerates towards the end. This increases the average speed of the motor, while minimising the chance of skipping steps.

The following sequence is used to move each tile. First the robot is moved to the available tile (the X, Y and θ axes are stepped concurrently, with the θ axis adjusted to align with the orientation of the tile). Then the Z axis lowers the gripper onto the tile and the vacuum is activated. The Z axis then lifts the tile. The robot is then moved to the target position and orientation (again X, Y and θ are stepped concurrently). The Z axis is lowered, the vacuum switched off, and the Z axis raised again. Once all the tiles have been moved, the robot manipulator is returned to the origin.

D. Game state machine

The game state machine mediates play between the human and robot player including managing the calibration process, the starting of a new game, and detecting game completion. During play, the sequence described in the previous paragraphs is stepped through using the image analysis results and the game representation to control the transitions.

V. Results

The FPGA based smart camera performs all of the analysis in the images streamed from the camera in real-time (15 frames per second at 5 megapixels resolution). The results are then used to autonomously control the Trax playing robot.

Acknowledgment

The assistance of Ken Mercer and Colin Plaw in constructing the physical robot is acknowledged.

References

[1] D. Smith, *How to play better Trax*. Christchurch, NZ: David Smith, 1983.

[2] D. G. Bailey, *Trax strategy for beginners*, 2nd ed. Palmerston North, NZ: D.G. Bailey, 1997.

[3] D. G. Bailey, K. A. Mercer, and C. Plaw, "Autonomous game playing robot," in *International Conference in Autonomous Robotic Agents (ICARA 2004)*, Palmerston North, NZ, 2004, pp. 295-300.

[4] D. G. Bailey, K. A. Mercer, and C. Plaw, "Computer games: Out of the `box' and onto the table," in *Tenth Electronics New Zealand Conference (ENZCon'03)*, Hamilton, NZ, 2003, pp. 151-156.

[5] Terasic, *TRDB-D5M 5 Mega Pixel Digital Camera Development Kit* Version 1.2: Terasic Techologies, 2010.

[6] Terasic, *DE0-Nano User Manual* Version 1.7: Terasic Technologies, 2012.

[7] B. Funt, K. Barnard, and L. Martin, "Is machine colour constancy good enough?," in *5th European Conference on Computer Vision (ECCV'98)*, Freiburg, Germany, 1998, pp. 445-459.

[8] D. Levy, *Computer gamesmanship*. London: Century Publishing, 1983.

[9] D. G. Bailey, "Trax tips - equivalence principles," in *On Trax*. vol. 8: The New Zealand Trax Association, 1998.