Baxter plays TIC-TAC-TOE while shopping with Simtrack

Michael C. Welle, Nadine Drollinger

September 18, 2017

Abstract

This is the documentation serves as instruction manual for the project Baxter plays Tic-Tac-Toe while shopping with Simtrack. This project was developed from 17/07.2017-29/09/2017 at Hong Kong University of Science and Technology under the supervision of Prof. Michael Wang & Prof. Hang Kaiyu. In the following we not only explain how to let Baxter play but also how to set up and operate the demo and provide an insight into the software, its underlying structure and architecture.

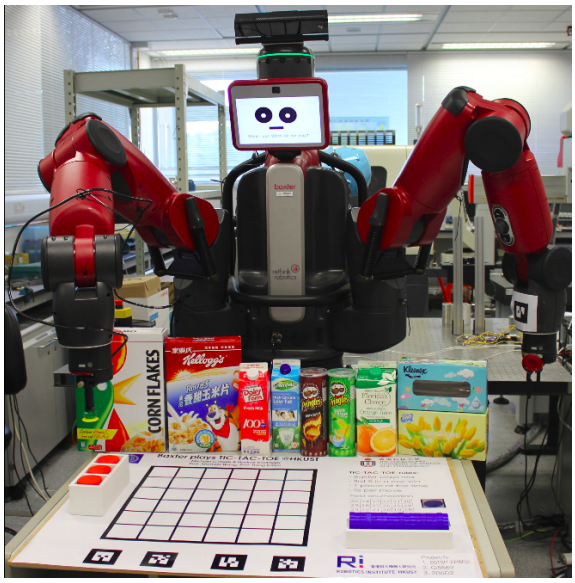


Figure 1: Baxter plays TIC-TAC-TOE and shopping

1 Quick start-up guide

The following instructions and set up descriptions describe how to set up Baxter in order for to play TTT.

1.1 Hardware set-up

The following hardware is necessary:

- workstation
- Baxter robot
- kinectv2 mounted on top of Baxter's head
- web cam mounted on Baxter's chest
- Baxter vacuum gripper mounted in Baxter's right arm!!!
- compressor
- game board fixed on game plate
- 36 game pieces (18 red and 18 blue)
- 2 storage boxed for the game pieces
- table
- shopping items
- shopping basked

1.1.1 How to fix the game board?

If you need to use a new game board you can download the pdf.file from the github https://github.com/HKbaxterteam/Baxter2017/tree/master/others. When printing it is essential that you choose the option 'actual size' in the printer settings! DO NOT choose 'fit size' as then the cells shrink by 1mm and you would have to adjust Baxter's offset by hand!
To test if the game board was printed correctly, just measure the cell size. It must be 60mm in each direction. If it is 59mm you have to reprint it!
The game board has to be fixed on the provided wooden plate. To fix the piece boxes, cut out a hole in within the gray area and put double sided glue tape in it. Its important to fix the boxes on the game board, because if for some reasons Baxter hits the piece box it would otherwise destroy the game field!

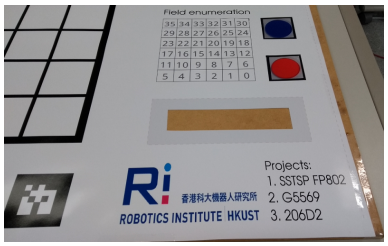


Figure 2: how to cut out the area for the glue tape

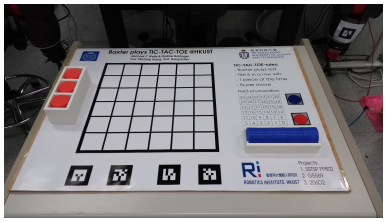


Figure 3: how to place and fix piece box

1.1.2 How to place the game board and table wrt Baxter?

The game board and the table should be placed in front of Baxter roughly as shown in the picture. Its is important to check that the AR-tags can be seen by the kinect.

1.2 Check-list before playing

Make sure the hardware setup is properly done and double check the following points:

- the vacuum gripper is correctly (!) mounted on Baxter's right arm
- the vacuum compressor has to be connected to the vessels
- the compressor is connected and set to a pressure of 6bar
- the game board is placed in front of Baxter within a distance of approximately 13-17 cm from Baxters torso, compare with picture XX

- the area around Baxter has to be free of obstacles
- the storage boxes for the game pieces are mounted on the marked area on the game board
- Baxter's game piece box is fully loaded, 3 stacks with 6 game pieces each
- Baxter gets 18 red pieces, the human opponent 18 blue pieces
- the emergency button is not pushed and ready
- Kinect is plugged into a USB 3.0 port.
- USB-cam is plugged into a different USB-bus as the Kinect.

Figure 4: vacuum gripper to be mounted on the right arm

Figure 5: Distance of table from Baxter

In the case that Baxter fails to pick up the game piece , Baxter will ask you to place it for him manually, see figure 6. Its important that you take it from the correct stack!

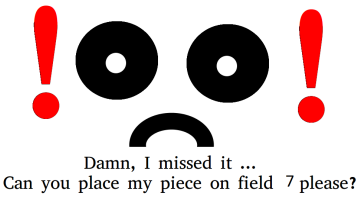


Figure 6: Baxter ask to place his piece on field number 7

1.3 Game preparations

1.3.1 Start Baxter

If the all the points on the checklist are ok then start the workstation and Baxter. Make sure the Ethernet cable from Baxter is connected to the workstation. Disable networking on the workstation and bring up a terminal.
Wait for the Baxter to show the Rethink logo and follow these instructions:

```
sudo ufw disable    # disables fire wall
sudo avahi-autoipd eth0   # claims ip-adress on ethernet 0
```

check that the claimed IP address is in the "catkin_ws/baxter_real.sh" file. Check that the hostname is "baxter7149".
Note! You only have to do this once, when taking over the project! To check the hostname of a Baxter connect a keyboard to it and press ctrl-alt-F3.
open a new Terminal and connect to the Baxter:

```
cd catkin_ws
./baxter_real.sh
```

To start everything at once do: (This is not recommended as it is very hard to find an error should one occur). To check if it was successful do a

```
rostopic list
```

If it was successful you should see all the available topics listed, if not , check if the Ethernet cable is connected, check if the firewall is disabled and if the terminal the ip-address was successfully claimed.

```
roslaunch game_master hkust_baxter_demo.launch
```

After everything has started correctly, you can use the GUI to control the game.

In order to start the three parts separately open 3 terminals which are connected to the baxter and run:

```
roslaunch game_master baxter_moveit_ttt.launch
```

The robot should be enabled (joints get current) and rviz starts with manual moveit configured. If there is a need to move the robot manually do it here. If not close rviz to save resources.
Next start the Tic-Tac-Toe part:

```
roslaunch game_master game_prep.launch
```

An rqt-gui should open and show an integrated rviz on the right side. The left side is filled with two images that are not yet shown.
Next start the Simtrack:

```
roslaunch game_master simtrack.launch
```

After some time (can be up to 30 secs) the output from the simtrack can be seen on the left side of the gui.

1.3.2 rqt gui

The rqt-gui can be seen in Figure 7. It shows the usb camera result when detecting the TTT game board at the top left.
To the right of this output is the Baxter dashboard which is used to control the TTT game. Right below it is the HD-color image output from the kinect.
On the lower left we can see the output of the simtrack which shows tracked objects in green.
Right next to it the simtrack shopping cart is displayed and can be cleared with the clear button.
All the way to the right we see the Baxter in the moveit configuration.

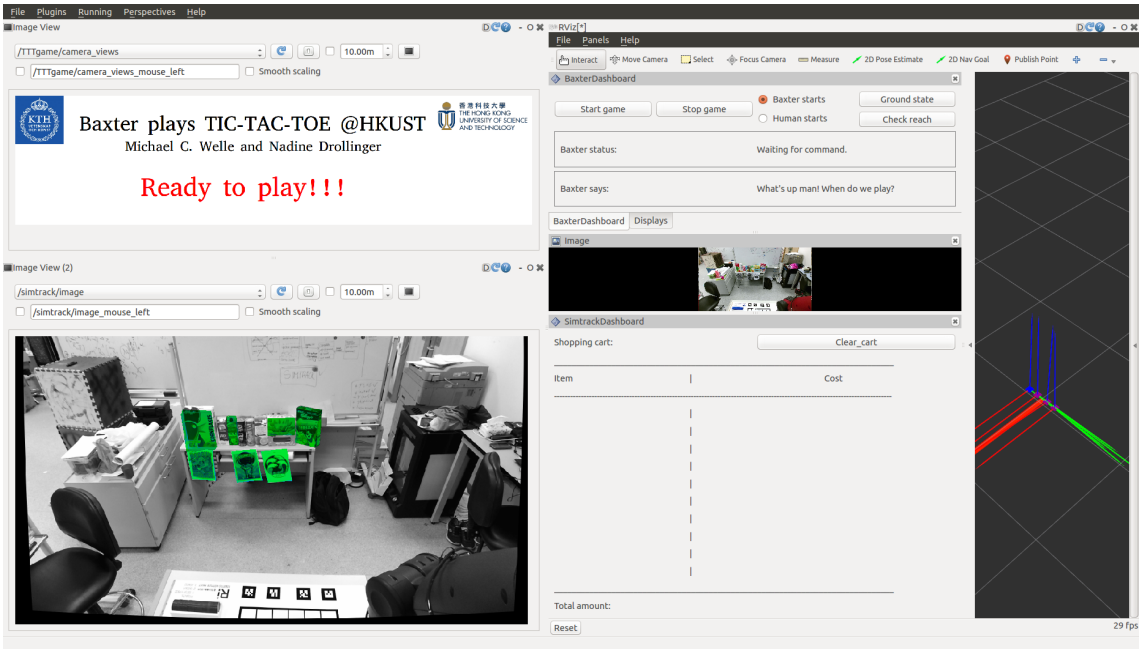


Figure 7: The rqt gui for controlling the demo

After everything has started correctly, you can use the GUI to control the demo.

1.3.3 setup check

Before starting the game or any function make a quick check-up:

- Is the game board visible for the usb camera? Compare it to figure 7?
- Does the kinect have a good view of the 4 Ar-tag markers (visible in the gui and compare to figure 7). Also the terminal should not output a warning that it doesn't have 4 markers.

1.3.4 moving to ground state

Use the button "ground state" to move the right arm to the default position.

1.3.5 Reachability check

Before you can actually play, you need to make sure that the game board is within Baxter's reach. Therefore use the button "check reach" and Baxter will try to reach the four corner cells. If you have placed the game board in the correct position Baxter will be able to reach all 4 of them. If Baxter cannot reach all of them, adjust the board position, wait for the ar-pose to notice the change and try again.

1.4 TIC-TAC-TOE The Game

TTT is a simple two player game where each player gets assigned a color and 18 game pieces.
The game is played on a 6-by-6 game board. The goal is to get as fast as possible 5 game pieces in a row, no matter in which direction. It can be horizontal, vertical or on the diagonal. The player that achieves this first wins.
The rules for the game are the following:

- Baxter is always RED
- it is randomly decided who places the first game piece
- each player places 1 game piece at the time
- game is start via GUI
- position of game board and storage box is as described
- the human has 5 seconds to do his move

Choose the starting player with the radio buttons in the gui (Baxter is the default).
Press "start game" to begin playing.

1.4.1 Game-flow

The entire game-flow can be seen in figure XX. The quick version is as follows:

1. Baxter moves to the piece box, picks up a game piece and places it on the board.
2. As soon as he releases the piece the usb-cam starts to detect it (this will only succeed when the arm is moved out of the view)

3. As soon as Baxter detects it's own piece and confirms that it was placed correctly, the timer for the human of 5 seconds starts.

4. If Baxter fails to place his piece he will ask the operator to place it (It will show up on his face)

5. The human is allowed to place the piece as soon as Baxter released his own even before Baxter detects his own piece. From the time Baxter detects it the human has 5 seconds to place his own piece and free the game board from his hands.

6. While the human is placing his piece Baxter picks up the next piece and waits.

7. As soon as the detection of the human piece is successful, Baxter will make the next move.

1.5 Simtrack shopping

The GUI provides a basked view of items and prices as well as the total price.
In order to buy an item one needs to move the item from right to left when facing Baxter. This will only work with designated objects as well as only when tracked the entire time.
In order to remove items from the basked one can click the "clear cart" button or move the item from left to right when facing Baxter.
It is only possible to shop for max 10 items.
It is possible to buy the same item multiple times.

1.6 Common pitfalls

If Baxter does not work as you expect him to, here is a list of common mistakes that were encountered during the project:

- **Baxter did not get enabled when launching the demo** Make sure you are correctly connected to the Baxter in your terminal via the baxter_real.sh script. Do this for all terminals!

- **Robot semantic description not found. Did you forget to define or remap "/robot_description_semantic"?** Make sure the terminal running game_prep is connected to the Baxter.

- **usb-cam fails** from time to time the usb-cam will not start (the window that opens by launch is fully gray). Unplug and replug the usb cam (you can use a external program like "Cheese webcam both" to check if it is working.

- **USB-cam still fails** Check the usb bus with "lsusb" if it crashes restart the machine.

- **Terminal shows: "Not 4 Markers" warning** check if the 4 markers are in the view of the Kinect (they need to be in view of the pointcloud).

- **Baxter hits the corner of the box** make sure the box is correctly positioned at the game board, adjust if necessary.

- **Baxter misses the box or field** make sure the tf from the Kinect to Baxter is still correct(If you moved the Kinect you need to readjust). Also the angular correction is dependent on the quality of the ar-tag readings (if the board is parallel to the baxter is best).
- **Baxter picked up a piece but doesn't move** Most likely the target is outside his reachable space and he can therefore not find a solution how to get there. He will try 3 times and then continuous as if he had managed it. During game place Baxters piece to continuous playing. When the game is over use the check reach function to make sure Baxter can reach the entire field.
- **Baxter did not suck up a piece** This can happen from time to time as to the tolerance of the positioning. Baxter will notice it when he looks for his piece and ask the Human to place it. !! take the piece from the stack Baxter wanted it from!!!.
- **Simtrack does not detect my object** Hold it still for a while to improve detection.
- **Simtrack loses track when I move left or right** Move slower.

1.7 Debugging with GUI

The Gui shows different outputs that can be used to debug wrong behavior of the baxter without going into the shell output.
The Gui shows the camera input in 4 seprate views (shown in figure 8.

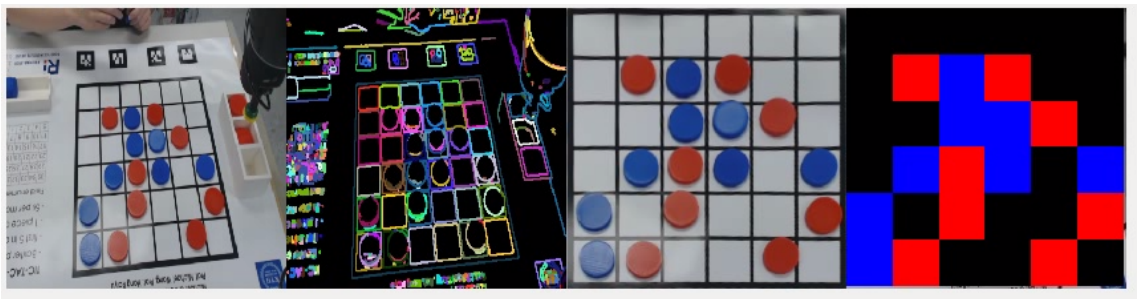


Figure 8: Camera output when detection is successful

The most left image shows the input the camera node revives, the next shows the output of the canny edge detector and shows all the detected contours. The next image is the projective transform of the cut out game board contour(the largest detected contour).
The image all the way to the right shows the game board how Baxter detected the pieces (black=empty, red =red piece, blue=blue piece) in the respective positions. This is the game board Baxter's AI will perform a move on.

If the detected larges are is not large enough the game board detection will fail. This happens when the arm is moving in front of the usb-cam or when the human places it's own piece.
The resulting output can be seen in figure 9.

Figure 9: Camera output when detection is not successful

The first and second image is the same as before but the third images shows the detected larges area and gives a clear indication why the detection failed. It is normal for the detection to fail while Baxter or the human perform a move.

The Baxter also gives out tow statuses into the GUI, Baxter status and Baxter says. Those messages can be used to determent where in the program Baxter is and should Baxter stop to do anything it gives an indication for what he is waiting. An example output can be seen in figure 10.

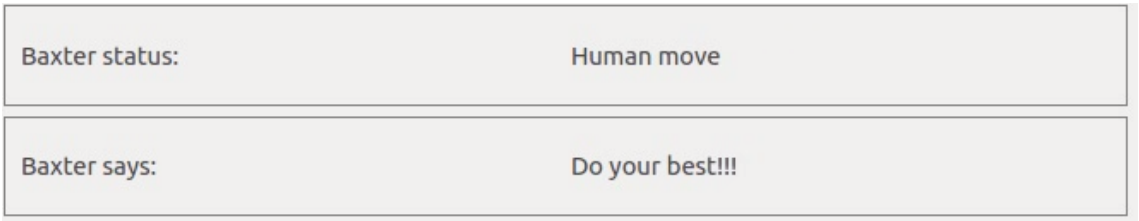


Figure 10: Baxter status output.

All the way to the right in th Gui the output of the AR-tag detection is shown. It is important that it displays 4 individual markers as well as one resulting big red arrow. This arrow is the Ar-tag pose he assumes and uses as basic for his position calculations. Make sure that the arrow is between the tags as shown in figure 11.

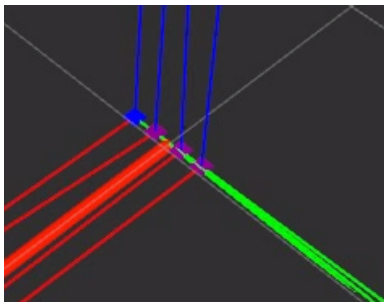


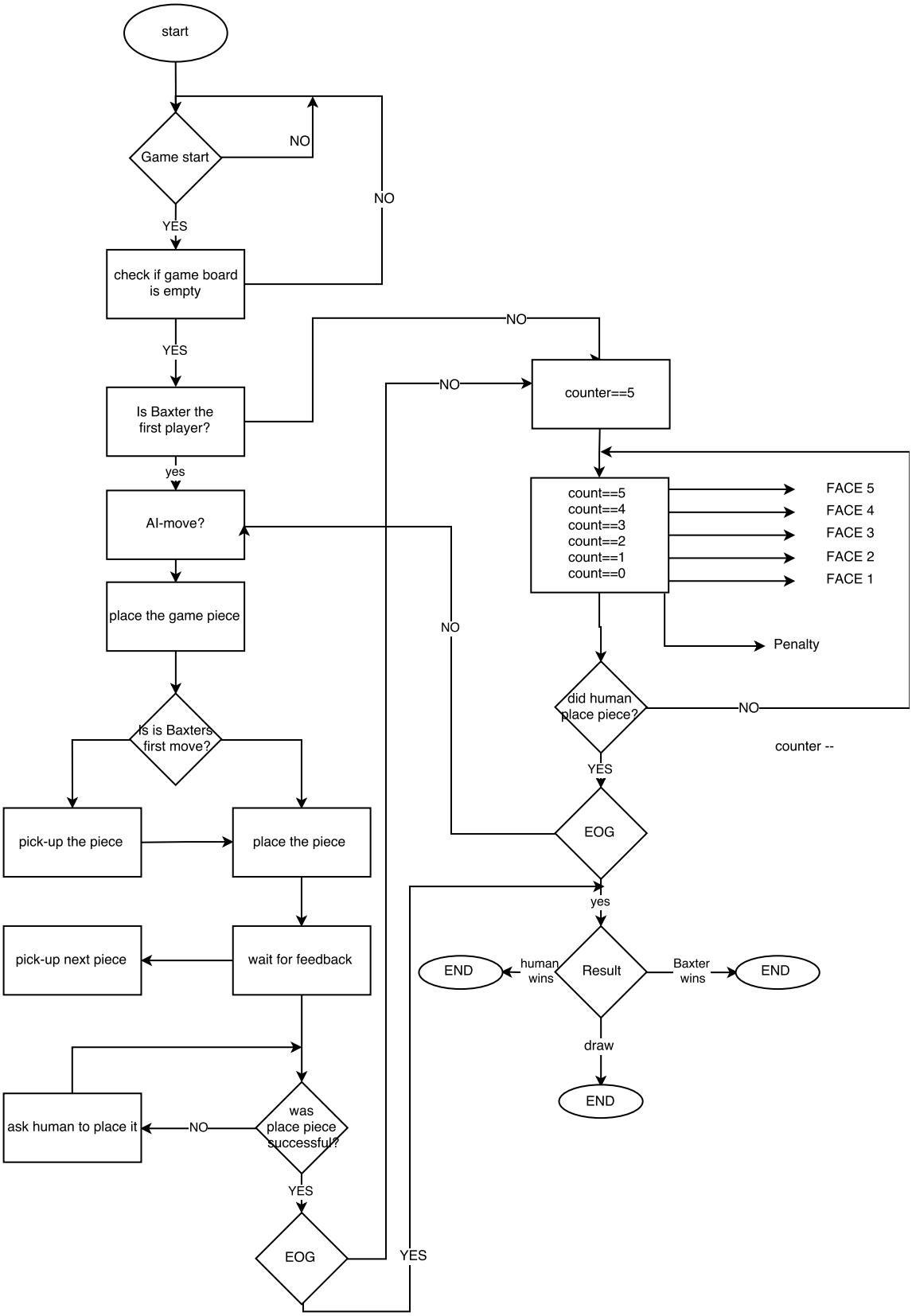
Figure 11: Ar-tag detection and resulting pose

2 Introduction

Inn this project the Baxter robot was used to play TIC-TAC-TOE against a human on a 6x6 game field. Baxter uses 2 different cameras. The web cam mounted on his chest to detect and observe the game board and the kinect mounted in his head to get the pose (position & orientation) of the game board. The underlying AI enables Baxter to play and win. A draw is also possible. The aim of this project was to get the Baxter robot to play TIC-TAC-TOE against a human on a 6x6 field.
The graph on the next page shows the structure of the program and the flow.
Simultaneously it is possible to conduct shopping using the Kinect and Simtrack.

Figure 12: Baxter behind the gameboard

2.1 Gameflowchart



3 The project setup

3.1 Hardware

- Baxter robot
- kinectv2
- webcam
- Baxter vaccum gripper
- vaccum compressor
- gameboard (designed by us)
- shopping items (Pringels(green),Pringels(brown),Kellogs(tiger),Kellogs(normal), milk,Kleenex,Kleenex(yellow),BearSweets,Juce).

3.2 Software

The entire project is realized in ros and uses opencv for the vision part.
Follow the setup.txt file from the git: https://github.com/HKbaxterteam/Baxter2017.
The main parts are:

- moveit (https://github.com/ros-planning/moveit_robots.git)
- baxter sdk (http://sdk.rethinkrobotics.com/wiki/Simulator_Installation)
- kinect2 bridge (https://github.com/code-iai/iai_kinect2)
- simtrack (https://github.com/karlpauwels/simtrack)

The Baxter2017 repo contains the following nodes:

- ai
- grasping
- game master
- camera
- gui
- simtrack_gui

The communication is done via actionlib and a sketch of the interaction between nodes can be seen in figure XX.

3.3 Limitations and assumptions

The following hard facts should be kept in mind when adjustments are done:

- Baxter always plays red
- Player numbers: blue= 1, red= 2, empty cell = 0
- The Pick up pose as well as the pose of the fields relay of the pose of the Ar-tags.
  The accurate placement only works when the printed out game board in 100% size is used.
- The simtrack as well as the simtrack_gui will only recognize the specific objects.
- The Baxter assumes that the player does not cheat.

3.4 The game-master node

The game master node is the heart of all operations and the ros actionlib package is used to control the interaction of the nodes and preempt tasks if necessary.
The following can be changed in the game master if required:

- faces of Baxter
- length of countdown for human.
- status massages send to the GUI.
- sleep times between steps.

3.5 The AI-node

To let Baxter play in a smart way again humans the mini-max-algorithm with $\alpha - \beta$pruning is used. The search tree created by the mini-max-algorithm is shortened by pruning which means cutting of the irrelevant branches of the search tree. The depth is set to 4 as afterwards its get too slow. The heuristic knows all the winning moves in the game board:

```
1  // 12 horizontal winning
2          {0, 1,  2,  3, 4},
3          {1, 2,  3,  4,  5},
4          {6, 7,  8,  9,  10},
5          {7, 8,  9,  10, 11},
6          {12,  13,  14,  15,  16},
7          {13,  14,  15,  16,  17},
8          {18,  19,  20,  21,  22},
9          {19,  20,  21,  22,  23},
10         {24,  25,  26,  27,  28},
11         {25,  26,  27,  28,  29},
12         {30,  31,  32,  33,  34},
```

```
13          {31,  32,  33,  34,  35},
14   // 12 vertical winning
15          {0,  6,   12,  18,  24},
16          {6,  12,  18,  24,  30},
17          {1,  7,   13,  19,  25},
18          {7,  13,  19,  25,  31},
19          {2,  8,   14,  20,  26},
20          {8,  14,  20,  26,  32},
21          {3,  9,   15,  21,  27},
22          {9,  15,  21,  27,  33},
23          {4,  10,  16,  22,  28},
24          {10, 16,  22,  28,  34},
25          {5,  11,  17,  23,  29},
26          {11, 17,  23,  29,  35},
27   // 4 main diagonal winning
28          {0,  7,   14,  21,  28},
29          {7,  14,  21,  28,  35},
30          {5,  10,  15,  20,  25},
31          {10, 15,  20,  25,  30},
32   // 4 other diagonal
33          {1,  8,   15,  22,  29},
34          {6,  13,  20,  27,  34},
35          {4,  9,   14,  19,  24},
36          {11, 16,  21,  26,  31}
37    };
```

The scoring of the heuristic checks how many pieces of each color are in a row and scores the points accordingly.
Here we use the logarithmic steps of $\{0, 1, 10, 1000, 100000, 10000000\}$.
For 1 piece (of one color) in a row it scores 1 point. For 2 pieces (of one color) it scores 10 points, for 3 pieces in a row it scores 1000 and so on.
As soon as one opponent piece is in a row it scores zero points. This is done for all rows and the respective scores are then summed up.The move with the highest score is then picked and executed. If there is more than 1 best move Baxter picks one move out of the set of best moves randomly.
As a endgame can often lead to a draw which gives a score of 0 in the heuristic Baxter would basically play random as he assumes any attack he would make would be blocked anyway. To counter this behavior a check is included that if every move has the same score it researches the tree but with a lower depth => therefore entering a more greedy behavior and more human.
The following can be changed in the AI node if required:

- depth of alphabeta search
- scoring for heuristic

3.6 The grasping node

The grasping node is home to all the moveit components as well as the environmental setup for the planning interface. The grasping node subscribes to the ar-tag

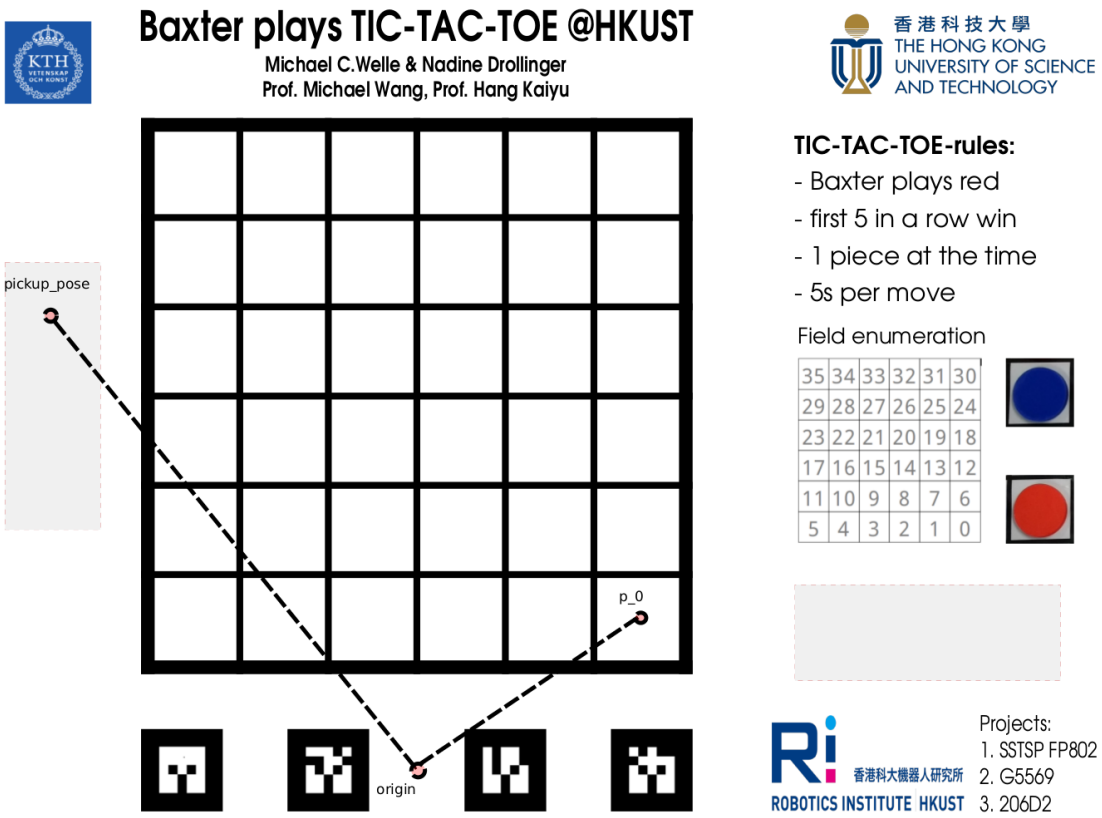


Figure 13: game board showing the origin, the pick-up-pose and p0

pose publisher and calculates the average x,y and yaw position of them. These parameters are set in the grasping node with "offset_p0_pose_x=-0.110"and "offset_p0_pose_y=0.161" as well as the pick-up pose "offset_pick_up_pose_x=-0.335" and "offset_pick_up_pose_y=-0.2635".
Given this position (in the middle of the 4 ar-tags) and the absolute real world offsets in x and y towards the first pick up stack as well as to the field number 0, those positions (pick up pose, field 0 pose) are calculated.

The grasping node follows a pick and place routine and tries to reach every position at most 2 times before skipping them and continuing. This gives a good robustness sense as the game can continue with a little help of the operator.
The grasping node sends a feedback message to the game master when the piece is placed so the camera node can start looking for the piece while the grasping continues and picks up the next piece.

To avoid collisions with objects in the direct environment with Baxter, collision models for the table, the web cam and the kinect were added. The following can be changed in the grasping node if required:

- real world offsets from ar-tag pose to pick up- and field 0 -pose.
- Planner can be changed (current: RRTConnect)
- tolerances for the moveit group
- number of planning attempts as well as planning time.

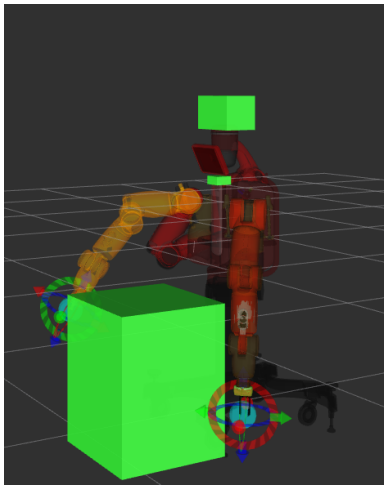


Figure 14: Collision models for table, webcam and kinect

3.7 The camera node

The camera node consists of two parts:

1. board_cutout
2. camera

3.7.1 1. board cutout

The board cut-out node is a simple pre-filter node that cuts the raw image provided by the usb-cam in a more suitable format.
The following can be changed in the board_cutout node if required:

- change the cut out position, length and width.

3.7.2 2. camera

The camera node is works based in the image provided by the board_cutout. The camera node tries to find the largest contour in the given image with the help of the canny-edge-detector.
From the input image of the real game board and the outer contour of it, a homogeneous transform is performed to get a plane view of the game board. The result is a square game board without any angular displacement or scaling where the all the cells have the same size. In this new game board the game pieces are detected by looping through the regions of interest, detecting if they are occupied by a red or blue game piece of if they are empty. Depending on their occupancy state they then get a value assigned respectively. RED=2, BLUE=1, EMPTY=0;

The result is then a game board array filled with the values for blue and red. This game board can be seen in the GUI and is send to the game master.
The following can be changed in the camera node if required:

- number of rows and columns of the game board.
- difference threshold (for detecting red/blue/nothing)

- Size of the image that needs to be enclosed by the largest contour(this is to avoid getting something different then the game board).

3.8 The gui-node

The gui-node provides the Rviz-plugin "BaxterDashboard" which is used to control the Tic-Tac-Toe game. The plugin depends on a action server that is provided by the game_master node. Therefore rviz will not start or stop responding if the BaxterDashboard is activated while no game_master is running.

The gui node communicates with action lib and a simple command structure:

- command "1" is game start (send by the "start game" button.
- command "2" is game stop (send by the "game stop" button.
- command "100" is the special command to move the Baxter to the defined ground state. (send by the "ground state" button)
- command "200" is the special command to check if the Baxter can reach the outer fields. (send by the "check reach" button)

The following can be changed in the gui node if required:

- Layout and function of buttons.

3.9 Simtrack

Baxter can track the following items:

Figure 15: Baxter is able to track these objects

3.9.1 How to create the Simtrack models?

To create the Simtrack models you need a camera (we encountered better results with an SLR camera), you objects, and account in https://recap360.autodesk.com/ to create the models and one on https://sketchfab.com/.
The first step is to take pictures of the object. Therefore place the object on a table and go around and take pictures from all around the object while keeping the height constant. Then do the same thing again but lower the camera and go again around and take pictures from every direction. For our models we took approx 40-50 pics

per item. The free- trial version of recap360 allows max 50 pics per item. When you have the pictures you just have to upload them on recap360, wait for 5-15min and then you can download you model.

add pictures and simtrack ref

As this model contains not only you object but also some environmental parts, open in in mesh lab and size it down until only the object is left. Then compress the folder of the object by zipping it and upload it on sketchfab. After the folder is uploaded, name the project and then go to the 3D-settings. Remove the shade and set the sharpness to 20 as recommended in the simtrack manual, then download the zip.file. This zip.file now has to be unpacked and then go into the source-file and unpack until you retrieve the obj.file.

Then open this file in blender and add a plane to the object. This is necessary as the model has a big hole otherwise. So add a plane, scale, rotate and translate it until it fits. Then select the object and use the boolean modifier, set the solver to carve and mark the plane. Press apply and remove the plan. Your object should now be closed. When this is done you still have to scale your object. Therefore mark the object, set the units to meters and press n. Then use the scaling until your model has approximately the size of the real item. Then press crtl-shift-alt-c and select the center of gravity as the origin. For saving the file you need to export it as on .obj and then check the faces... Finally you need to go to the folder that you downloaded from sketchfab and copy the tex.jpg file and replace the one in your final folder.

3.9.2 Simtrack Gui

The simtrack shopping cart is controlled by the simtrack_gui node and is realized as a Rviz plugin.
in the simtrack node the name as well as the costs for each item can be set. The current configuration allows up to 15 objects simultaneously. In order to add an Object follow the "how to create Simtrack models tutorial" and extract the .obj, .mlt and the texture file in an folder that has the same name as the .obj and .mlt files.
Next run the following command (with adjusted path):

```
rosrun interface cmd_line_generate_sift_model pwd‘/src/simtrack/data/object_models/
```

In order to generate the SIFT features.
After generating the sift features the object needs to be added to the config file. Go to

```
simtrack\simtrack_nodes\config\demo_objects"
```

and add the folder name to the list.
Now the Simtrack will publish a node under the " \ simtrack\NAME" topic.
In order for the shopping card to recognize the movement and add or deduct the item from the card the item needs to be added to the code. Look at the implementation of the 3 demo objects (the turtles) and either rename them to the desired object or copy the functions in order to add the object.
The following can be adjusted in the simtrack_gui node:

- Name and costs for the objects.
- Threshold for adding or deduction the objects from the card (default set to 0.3m)
  adjust the variable "consec_needed" and "size_of_fields" respectively.

3.10 Baxter's faces

To make Baxter more human-like, 17 Baxter faces were created. By quickly switching between some of them a movement is created.If the human acts too slow a countdown of 5s is started.

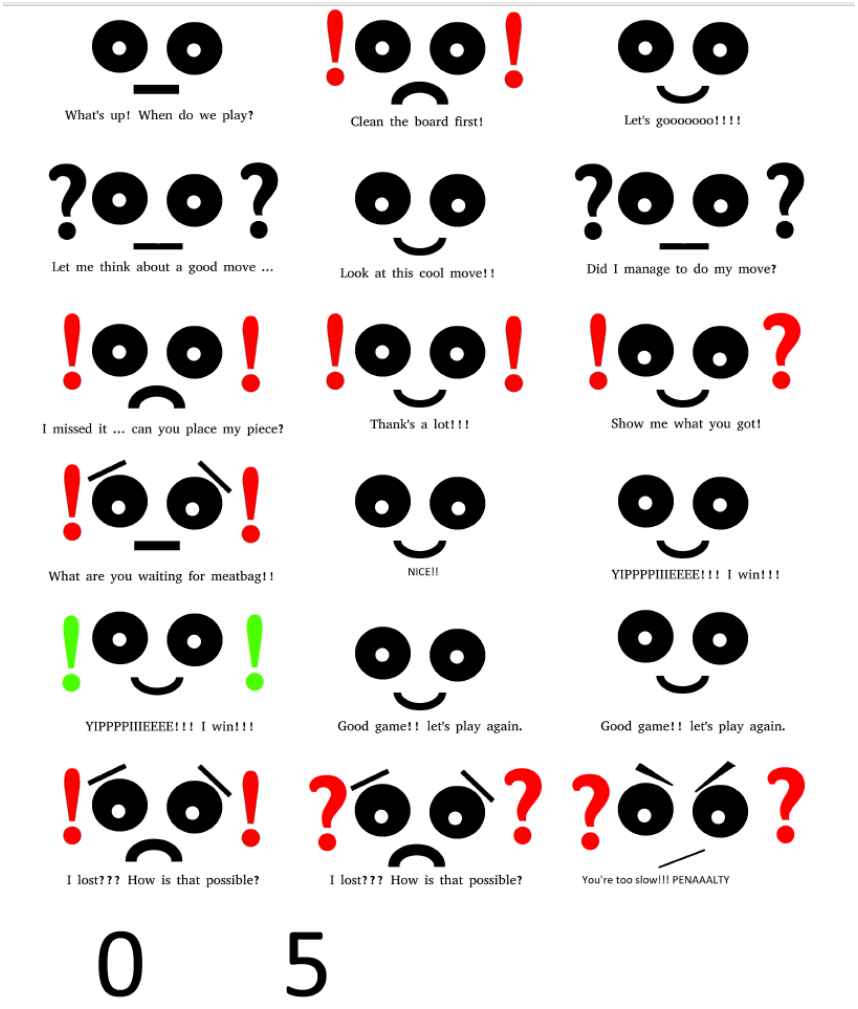


Figure 16: Baxter's faces overview

3.11 The game board

The game board designed for the TTT has the following properties:

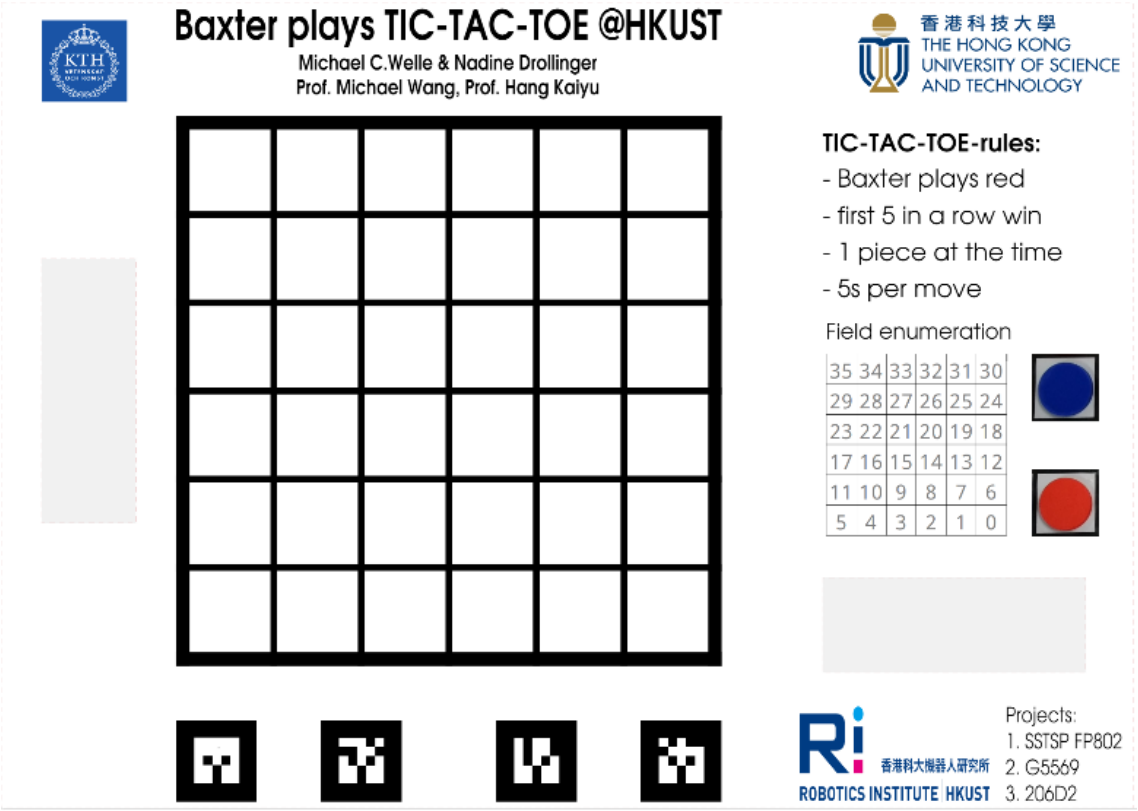


Figure 17: the game board

- 6-by-6 cells
- cell size = 60mm x 60mm
- inner frame thickness = 5mm
- outer frame thickness $10cm$
- total game board size =405mm
- 2 storage boxes for the game pieces (compartment box for Baxter)
- 4 AR-codes
- the enumeration map and rules

3.11.1 The Board for Baxter

The game board is observed by the web cam and is used to detect the board and pieces and pushed into and array.

The game board is enumerated the following way, whereas P denotes the next player. Meaning if the board was empty, P contains the number of which player starts the game.

As it is assumed that Baxter always plays red=2. Figure shows the empty game board.

0	1	2	3	4	5	
6	7	8	9	10	11	
12	13	14	15	16	17	
18	19	20	21	22	23	
24	25	26	27	28	29	
30	31	32	33	34	35	P

Table 1: game board model

The Array looks as follows: $\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & ... & 35 & P \end{bmatrix}$

3.12 AR-node

The AR-node uses the ar-alvar-package and the kinect to calculate to pose of the game board. As there are 4 ar-tags the result is 4 poses on which a moving average filter is applied to get a steady pose. This pose is then fed back into the game master.

3.13 How to determine the tf-transform from Baxter to the Kinect

To get the tf-transform from the Baxter to the Kinect the "baxter_h2r_packages" calibration package is used(https://github.com/h2r/baxter_h2r_packages/tree/indigo-devel/baxter_kinect_calibration). As we are using the kinect 2 make sure to check out the forked repository in the HKBaxterteam organization.

Follow the calibration read-me and copy the resulting tf-transform to the tf-tree launch file under the grasping_baxter node.
The tf transform will not necessarily be as accurate as it needs to be, there a two ways to improve the accuracy of the transform:
1. use bigger ar-tags to increase the accuracy. In order for the transform to adjust to the new size the size needs to be specified inside the:

`baxter_h2r_packages/baxter_kinect_calibration/launch/baxter_bundle_calibration.laun`

and adjust the "marker_size" parameter.

2. Tape another ar-tag to one of Baxter's arms and move it in front of him so the kinect can see it. Now adjust the transform manually and compare the resulting point cloud with the planning model of moveit. If they overlap perfectly the transform is done. You can also check if the pick up pose and place-pose works and adjust the tf-transform accordingly.

3.14 Appendix

3.14.1 Software-Packages

- our Github https://github.com/HKbaxterteam/Baxter2017.git
- ros::actionlib http://wiki.ros.org/actionlib
- ros:: MoveIt package http://wiki.ros.org/moveit
- AR-tags http://wiki.ros.org/ar_track_alvar
- Baxter SDK http://sdk.rethinkrobotics.com/wiki/Hardware_Specifications Baxter Joint names
- Baxter zero point http://mfg.rethinkrobotics.com/mfg-mediawiki-1.22.2/images/1/1e/1.1.2_User_Guide.pdf
- Simtrack http://www.karlpauwels.com/simtrack/