**Upload your project report in BlackBoard by 11:59PM, 12/03/2016**

**This project is a teamwork project (2 ~ 3 students of a team). You accomplish this project by working with one or two classmates. You can use any programming language and existing open source libraries to accomplish this project. Each team submit a project report and required files as indicated in the following tasks. List all team member names in the project report.**

**Project 2. Search and Robotic Vision**

**Task 1: Implement A\* search algorithm. Upload your source code and the final grid image with the shorted path in this project report. (30 points)**

**The following figure contains a robot, a door, and six chairs, each of them takes one grid. The figure also contains 10 by 10 = 100 grids. The robot need to avoid six chairs and find out a shortest path to the door. For each step of the A\* search, the robot can only move to one neighbor grid of 8 neighbors (east, southeast, south, southwest, west, northwest, north, and northeast).**

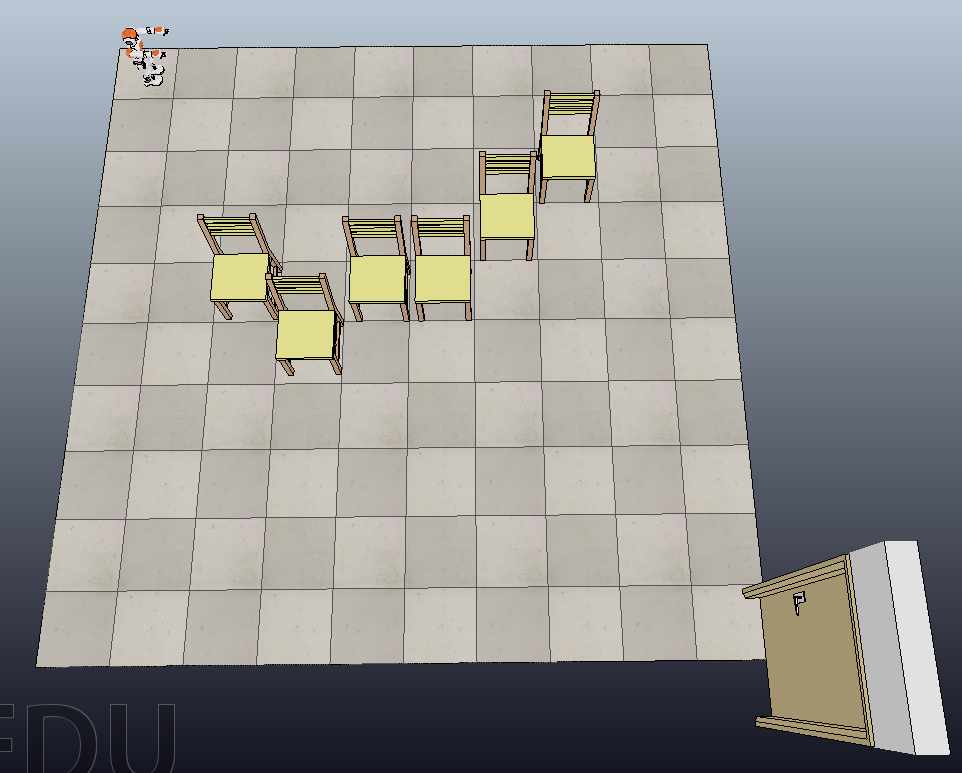
**You can reference A\* search algorithm in the PowerPoint file “PowerPoint\_Topic7\_Basic\_Search” (BlackBoard) and on the webpage as follows.**

[**http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html**](http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html)

**For reference code of programming languages of C++, Java, Javascript, Python, Objective C +, Lua, Ruby, C#, and Processing, you can reference the webpage as follows.**

[**http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html#code**](http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html#code)

**(Hint: can use a 10 x 10 matrix for modeling the 10 x 10 grids. Each element in the matrix is relevant to a grid in the picture. After the A\* search, visualize this matrix to show the shortest path.)**



**Task 2: A video file “Walk1.mpg” has been uploaded on BlackBoard. In order to track the moving people in the video, a preliminary python file “object\_tracking.py” (you can also use other programming language) has been uploaded on BlackBoard. The preliminary code has implemented the video frame extraction. All 612 frames will be stored on the current working space folder. You need to complete the following steps for moving object tracking in the video.**

**Step 1: read all frames (612 frames in total) (this step has been already done in the preliminary code)**

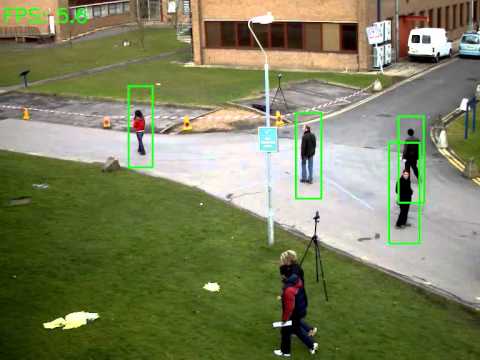
**Step 2: for each pixel position, calculate standard deviation along 612 frames**

**Step 3: set a threshold for standard deviation (the mathematical formulation of the standard deviation is in https://en.wikipedia.org/wiki/Standard\_deviation)**

**Step 4: based on threshold, extract the moving objects (remove static background pixels) and draw a rectangle onto the moving objects, save all frames.**

**Step 5: rebuild 612 frames to a video, in which moving objects are tracked**

**After the task 2, you video is similar to the following frame. (Note: the following figure is just a sample. The content of the video “Walk1.mpg” is different from the following sample frame.). Your program doesn’t need to**



**(Frame is acquired from:** [**https://www.youtube.com/watch?v=OvCYRrtfx-s**](https://www.youtube.com/watch?v=OvCYRrtfx-s)**)**

**Upload your object tracking video and source code on BlackBoard.**

**Task 3: Given a V-REP scene file (the file name is “vrep\_obstacle\_detection.ttt”. It is uploaded on BlackBoard. The scene picture is presented as following), implement a path following robot using the V-REP simulator. The current Pinoneer\_p3dx cannot directly follow the path in the scene. In order to build your motion planning path, you can reference the tutorial video:** [**https://www.youtube.com/watch?v=xI-ZEewIzzI**](https://www.youtube.com/watch?v=xI-ZEewIzzI)**. It shows the step-by-step implementation of modelling a line following robot.**

**When you simulate the V-REP scene, then you run the python program “python\_avoid\_obstacle.py”, the Pinoneer\_p3dx car can be driven by the program. The embedded ultrasonic sensors can avoid walls (but cannot avoid red obstacles). In order to track the path and move along the path, a camera sensor will be used to detect the red obstacles. Based on the vision sensor’s tracking on path, the purpose of the simulation is that the car can touch each red obstacle and then continue to move along the path until all 10 red obstacles are touched. The final simulation effect is similar to the following demonstration.**

[**https://www.youtube.com/watch?v=OtL96ebm3iQ**](https://www.youtube.com/watch?v=OtL96ebm3iQ)

**You can make reference of the following video for accompanish the task.**

[**https://www.youtube.com/watch?v=kOjQRYmeX\_o**](https://www.youtube.com/watch?v=kOjQRYmeX_o)

[**https://www.youtube.com/watch?v=SQont-mTnfM**](https://www.youtube.com/watch?v=SQont-mTnfM)

**Hint:**

1. **Reference the file “python\_avoid\_obstacle.py” to see how to get and control the left\_motor\_handle and right\_motor\_handle (by using APIs: simxGetObjectHandle and simxSetJointTargetVelocity).**
2. **To capture the image, use the API: simxGetVisionSensorImage.**
3. **Note that V-REP provides the image data as a contiguous array of n elements, and not a multi-dimensional array. So, basically, if you have an image of resolutionX and resolutionY, and you are transmitting a colored image (b=3), you have n=resolutionX\*resolutionY\*b, and you would access individual pixels (position (X;Y)) with:**

**simxChar pixelRed=image[b\*(Y\*resolutionX+X)+0];**

**simxChar pixelGreen=image[b\*(Y\*resolutionX+X)+1];**

**simxChar pixelBlue=image[b\*(Y\*resolutionX+X)+2];**

**Upload your scene file (“\*.ttt”) and source code files on BlackBoard.**

