Autonome systemen (2009 – 2010)

Practicum 4 – Aibo

Erick Wilts - #1635085
Matthijs Dorst - # 1380982

Contents

Introduction

With the rising of the modern AI, a great variety of path planning methods for mobile robots have been developed. One of the more popular methods is the vector field path-planning method, due to its mathematical simplicity and suitability for dynamic environments. It is like the robot moves through a field of forces attracting and repulsing it.

In the following text, we describe how an Aibo robot could be programmed to walk to the space between a green and a pink pillar, as if to find a bone that was buried between these pillars. Using a variation on the vector field method, the Aibo should be attracted by the green and the pink pillar (however be repulsed by them when coming too close) and repulsed by obstacles. Thereby, its path is guided by another landmark, the yellow pillar, if the green and pink pillars are invisible by obstacles or because of absentness.

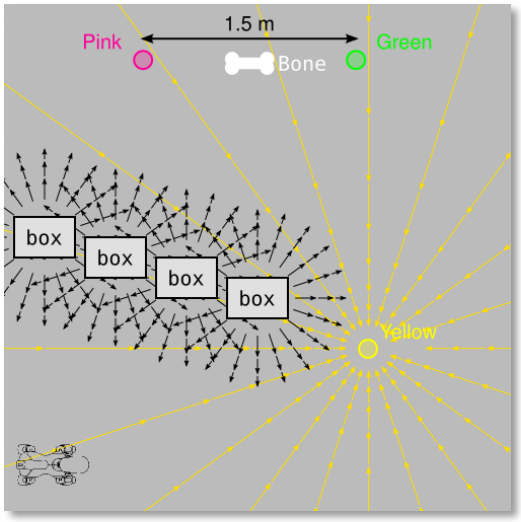

Figure 1: Experiment conditions (Courtesy of Drs. S. De Jong?)

The 'variation' part in this implementation for the vector field method is that the vector field is calculated every time the robot has walked a little bit, and that only one vector is calculated for the position of the robot, instead of the entire vector field, because there is no overview of the vector field if the perception only goes through the camera of the Aibo.

Model

Our model is based on three distinct states and cycles through them sequentially. These are, in order: observing (sense), planning and walking (acting). In the subsections we will discuss each state.

Figure 2 illustrates how the model cycles through each state until termination conditions are met. Note that without pillars at the designated relative positions the robot will continue to seek and walk indefinitely.

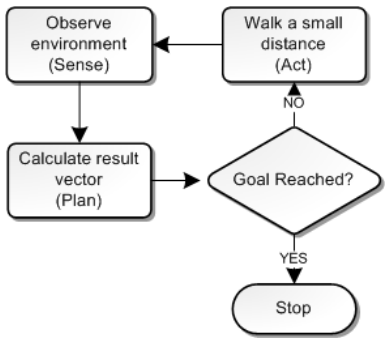

Figure 2: SPA Architecture

Observing

First of all, the Aibo has to know where it is. Therefore, it pans its head as far as it can, right to left, in small steps. During the head sweep, the positions of the pillars and the corresponding distances are calculated. If a pillar disappears from the camera image, the vector is not removed, but memorized, and overwritten when another blob corresponding to the pillars specific color map appears – though only when that blob has a greater apparent surface area. A large pillar can thus cause the robot to ignore a smaller pillar it saw first (which presumably is further away). Furthermore, the value of the 'distance' sensor (an infrared distance sensor in the chin of the robot) is measured every step of the sweep, so there is an overview of the objects surrounding it.

The distance to the pillars is estimated by its apparent location in the visual field: objects higher in the image will generally be further away, though the method becomes imprecise at longer ranges. Of

course, using this way of distance measurement requires the head tilt and the neck position of the robot to be constant during the sensor sweep, which we will see is not always the case.

Planning

The resulting vector represents the direction and the distance the robot has to walk. It is calculated by converting the vectors to x and y values, summing over these and converting the result back to vector format. Of course, the proximity vectors are not as important as the pillar sensors, if only because there are much more of them. A weight is added to the length of each proximity and pillar vector to compensate for this, and through trial-and-error weights are calibrated to prevent collisions while ensuring the robot reaches its target.

An additional part of the planning process is termination-checking: when the robot reaches its destination it needs to stop. Using an estimate for the location of unseen pillars and a relatively large margin for error the robot is instructed to terminate once both pillar vectors are presumed to be far enough apart.

Walking

A reliable and semi-precise method of movement is essential to following the determined path. Using the target vector calculated earlier, the Aibo robot is first instructed to turn into the new direction and consequently walk a distance proportional to the length of the calculated vector.

Implementation

Observing

As discussed, the robot observes its environment by setting the head pan to its maximum range of approximately 93˚ away from center. During our tests we noted that despite giving the Urbi command for a 93˚ headPan, actual pan is generally about 89˚ - but more on that in the results section. Over an arc of (in theory) 186˚ the robot pans its head from -93˚ to +93˚ in steps of 15˚ each. At each step the distance sensor measures how far away the nearest obstacle is. Additionally, the head pan sensor is used to measure the exact angle obtained as this can deviate from the angle the robot was instructed to attain.

In this way we obtain 13 vectors over a range of approximately 186˚ denoting distances and angles to all obstacles in range (insofar these fall not between measurements – a very thin pole could in theory poses a problem here). Additionally, the camera is constantly observing the visible area (at a speed of roughly 10 frames per second, though this we found to vary wildly) and processes each frame to find visible connected areas for three specific color ranges:

Color	Red (min / max)		Green (min / max)		Blue (min / max)	
Yellow	120	230	110	210	0	50
Pink	195	256	0	89	10	100
Green	50	100	100	256	0	50

Table 1: selected color ranges (from 0 – 255 maximum range).

If an area of at least 60 pixels is found, the coordinates, length, width and current head pan are stored for further processing.

Planning

Interpreting observations

For each vector, the relative x and y coordinates of the target object (be it pillar or obstacle) is calculated using:

$$x = \sin(angle) \cdot length$$
$$y = \cos(angle) \cdot length$$

Where angle and length denote the angle and length of the vector as relative from the robots head. For obstacles, the angle is the head pan at the moment of observation and the length is the distance measured by the distance sensor as fraction of the maximum range (150 millimeters). For observed objects in a specific color range, the angle (A) is calculated using the following equation:

$$A = \left(\frac{180}{\pi}\right)\left(headPan - \frac{x + \frac{width}{2} - \frac{\max width}{2}}{\max width} \cdot FieldOfView\right)$$

Here headPan is the current head pan (in radians) width is the width of the object in pixels, max width is the maximum width of the camera image (208 pixels) and FieldOfView is the view angle of the camera (56.9˚). This yields the angle in degrees to the object from the current direction (where straight ahead would be 0˚). The length of the vector (L) is used mostly relative and calculated as:

$$L = \frac{(\max height - (y + height)) - 40}{50}$$

Here max height is the maximum height in pixels of the camera image (160), y is the offset of the top of the object from the top of the image (in pixels) and height is the height of the object in pixels – thus we calculate the bottom of the object, subtract 40 and divide by 50 to obtain a decent estimate of distance. Note that the subtraction and division values are chosen as arbitrary constants found by a series of testing, though these do not translate into real distances. The subtraction is used to make pillars repelling once they come too close (40 pixels translates into roughly half a meter) and the division is used as a normalizing factor – far away objects appear to be on the horizon, which in the default standing position coincides with an offset of ~90 pixels from the bottom of the screen.

Data combination

We found the easiest method of combining the resulting vectors was to convert them to relative x and y coordinates (using the formula given earlier) and then simply adding these. Thus:

$$\vec{t} = \begin{pmatrix} \sum x_{pillars} + \frac{\sum x_{obstacles}}{N_{obstacles}/2} \\ \sum y_{pillars} + \frac{\sum y_{obstacles}}{N_{obstacles}/2} \end{pmatrix}$$

Here $x_{pillars}$ are the x coordinates of the pillars, $x_{obstacles}$ are the x coordinates of the obstacles and $N_{obstacles}$ are the number of obstacle vectors (13 in our case). That is, the averaged obstacle vector influences the resulting target vector $\vec{t}$ half as much as pillars do.

The target vector can then be converted back into a length (L) and an angle (A) using the simple equations:

$$L = \sqrt{x^2 + y^2}$$
$$A = \tan^{-1}\left(\frac{x}{y}\right) \cdot \left(\frac{180}{\pi}\right)$$

Here x is the total x of our vector $\vec{t}$ and y is likewise the total y of this vector. Note that L is here a dimensionless fraction (since all original vectors were normalized) and A is an angle in degrees.

Termination Checking

One of the more difficult aspects was to stop once the target position had been reached. Since the robot can only make accurate frontal observations (within a ~ ±200˚ angle) it would be impossible to detect both pillars simultaneously when approaching the target location at an angle. Instead we keep a record of the location of both pillars for one step and use this to augment our observations when a pillar is lost out of sight. Thus, if the robot first observes a green and pink pillar and after one step only observes the pink pillar, the green pillars relative position to the robot is calculated using its previously known location and the last performed movement.

Because movement is imprecise and distance estimates are inaccurate this provides only a very rough indication but more on that later. To estimate the location of an unseen pillar, we used the following (pseudo) code:

```
1     OldAngle =  pillar.angle  - walked.angle
2     y        =  pillar.length * sin(OldAngle)
3     x        =  Sqrt(pillar.length² - y²)
4     x        =  x - walked.distance
5     NewAngle =  -atan(y, x) * 180 / PI
```

The pillar.angle and pillar.length indicate the stored angle and length of the (unseen) pillar, while the walked.angle and walked.distance indicate the direction and distance walked since these values were stored.

After each iteration these stored values are refreshed – either by the new angle and length of the specific pillar, or by deleting them in case no new values are available.

As termination condition then we can now use the angle between the robot and both pillars: if this angle exceeds 120˚ (but not exceeds 240˚) we will assume the robot to be sufficiently close to a spot in between both pillars. If the pillars are 150cm apart, this means the robot will terminate once it is within approximately 25cm of the target location (though see results).

Walking

Using the Urbi commands for walking in a specific direction and distance (using robot.walk and robot.turn) the actual acting is a rather simplistic matter. However, we found that commands do not delay execution and additional commands are queued to be executed later. This presents us with the problem that the robot would walk, receive new commands while walking based on its current position, walk on, and act out the new commands only once the old command was completed – but

since the new command is based on the old location of the robot, it is generally inaccurate. What is worse, this process repeats itself so a once small error increases to the point where the robot would walk in a direction completely unrelated to its current position.

Since the first motor command executed after a walk command is the command to pan the robots head to the far right (at an -93˚ angle) and sensory readouts *are* continuous and current, we added a while loop after each walk command that would read out the current head pan and enter a sleep mode for 100ms if the current head pan was not at most 85˚. When walking, the head is set roughly straight forward (at a 0.0˚ angle), so only once the robot is done walking *and* done turning its head can the program stop sleeping and start observing again. Using this method, walk-command overlap was effectively eliminated.

Results

Observation results

We found several discrepancies in the observations as reported by the robot:

- The bottom limit of an observed obstacle was not constant at every angle for a constant distance. This was evidently caused by the head and neck tilt, which should be perpendicular to the ground yet proved to be at an about 80˚ angle. As a result, the camera image pivoted around the center and distance estimates for pillars further aside are less reliable than those for pillars directly in front of the robot.
- Distance estimates for obstacles sometimes suffer from a similar problem where the distance for an open range (exceeding the distance sensors maximum range of about 150 centimeters) would be reported as less than the maximum range. Most likely the robot measured the distance to the ground instead here. Since the problem only occurs for larger ranges and obstacles do not generate vectors until they are within a 1 meter range, this was not a major issue luckily.

Motor command results

Quite unlike for example the pioneer robots, the Aibo movements are unreliable at best. We found the following problems:

- A movement instruction of 30˚ would result in an actually moved angle of about 25˚. Similarly, for other values the angle would be generally undershot by sometimes as much as 15˚. We have added a 10% increase in motor command instructions for turns to compensate, which appears to be satisfactory.
- Walking distance instructions are similarly wildly inaccurate. An increase of 40% was required to make the robot walk the actually requested distance.
- Head pans have a theoretical range of (-93˚, 93˚). However, during testing we found the robot would sometimes stall if we put the sleep check at 90˚ as measured by the head pan sensor. As reported, setting the angle at 85˚ was necessary to ensure the robot would always be able to turn its head at least so far and continue to the next step.

General Results

Speed

Overall, speed is not something to look for in an Aibo. We measured the average time required to perform certain angles and distance movements. The results are shown in figures 3 and 4 respectively.

One interesting observation is that the time required to turn a specific angle is only loosely correlated to the angle itself – for example, it takes the robot longer to turn 40˚ than it does to turn 60˚! We have been unable to find a significant linear correlation between turn angle and time required, as should perhaps be suspected for quadruped locomotion.

Contrary however we did find a significant linear correlation for the time it takes to walk a specific distance, with the observation that for walking straight ahead an initial 'startup' time is required – see figure 3 for the correlation line and data.

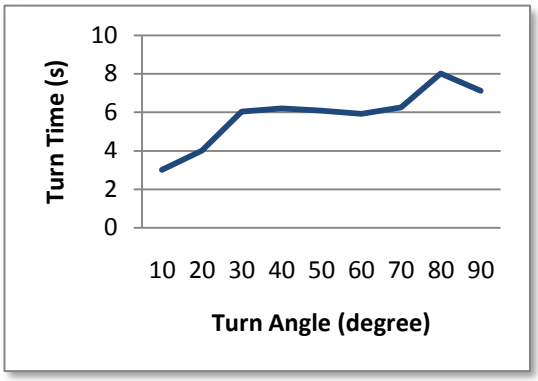

Figure 3: turn angle versus time

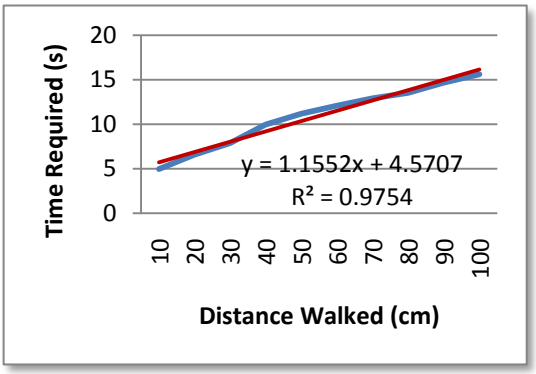

Figure 4: walk distance versus time

As should be clear, the initial time requirement for every movement is relatively large as compared to the additional time to move or turn further – about 3 seconds for turning and 4.5 seconds for moving. Thus, a 50cm walk only takes about 20% longer than a 25cm walk. To minimize the effect of this, we have limited our robot to relatively large steps: it will not turn for angles less than 10˚ and can move up to 1 meter per step. Naturally, increasing these values would improve even further but at the expense of accuracy. We found these values to be a fair tradeoff between the two. Still, the average speed is little over 1.5 meter per minute (when including observing) for an optimal situation and can be as little as 0.5 meter per minute when small steps are being taken in a crowded environment.

Accuracy

Because the steps taken by the robot are fairly large, the goal is only approximated to without a ~ 50 centimeter diameter. Since no mechanisms are in place to ensure equidistance from each target pillar and the termination condition is set to fire once the angle between pillars exceeds 120˚, accuracy is not extremely high. Furthermore, the chosen control architecture means the robot can overshoot the final step towards the target position. Decreasing the walk distance per step would improve this, but at the cost of speed.

Discussion

Purely considering our initial goal, to guide a robot with vector field path-planning, our implementation performs well. The most important and interesting discussion is whether we should use the search-plan-act-cycle. The tragic example of Shakey makes us happy that our Aibo only needs a fraction of a second to execute the planning phase.

Something that might need some improvement too is the drawing of the field vectors. Firstly because most of the time, the vectors towards the targets were too small. This could be because of the way the distance to a pillar is calculated. This could also be the reason for the fact we had to increase the walking distance with 40%. Secondly, the resulting vector seemed to be pointed in a random direction, however the length corresponded correctly with the distance that should be walked – either a bug in our implementation or the way displayed vectors are handled.

Of course, in theory, the termination condition should only be met if the angle between the pillars is 180° and both pillars are equally far away. In this perspective, our angle of 120° is by far not large enough, but then again, the robot could never be exactly in the middle of the pillars and there can never be great certainty about the position of invisible pillars. As mentioned, the robots approach circle for the bone position falls well within acceptable limits.

While an interesting problem overall, we feel the proposed architecture leaves something to be desired – after all, it is several decades old and abandoned for good reason. Furthermore the starting code was incapable of reading out a simple sensor and despite its multithreaded complexity failed to halt in between motor commands – adding the requirement for continues sensor measurements and sleep cycles to ensure no unwanted command queuing occurred.

To summarize, while fun to work on, the problem appears obsolete and the code foundation lacking in areas. The need for complex termination checking including the requirement of a semblance of a world model further hints at a control architecture best left in the past.