Interactions for evolutionary robotics simulator: collision avoidance and IR sensors models

Federico VICENTINI
Interactions for evolutionary robotics simulator: collision avoidance and IR sensors models

Federico VICENTINI federico.vicentini@polimi.it
† Politecnico di Milano, Italy

Abstract

This is just a first draft of a document that is supposed to describe in details the characteristics of a simple 2D s-bot simulator. This document will be subsequently revised by the author.
I. Introduction
The report gives a description of the specific modelling features used in an evolutionary robotics simulator. The functionalities presented are related to collision avoidance strategy evolution and proximity sensors simulation. The simulator itself implements the environment for evolutionary training of neural controllers. Regarding the interaction between the simulated robot and the simulated world, the assessment discussed provide contact and perception. The former principle is considered as a way of training the controllers to manage other objects in their space, the latter represents the supply of inputs for the controllers. On this basis the motion strategy and the overall behaviour of the agent is evolved and acted by the controller through the outputs (motors). Many motion functions are not described, the neural network coding is shortly outlined. The evolutionary robotics approach underlying the simulator is based upon the homogeneity hypothesis. Each agent is equipped with the same controller. The genotype coding the neural controller is cloned into each robot.

II. The robot
The robotic platform in use is the s-bot designed in the Swarm-Bot project. The robot is modelled as differentially driven circular robot. The model is bi-dimensional so each basic size is related to planar geometry (segments, diameters, …). The sensors mounting is also related to the robot geometry. The main quantities are reported in Table 1.

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>robot radius [cm]</td>
<td>5.8</td>
</tr>
<tr>
<td>collision distance [cm]</td>
<td>3.0</td>
</tr>
<tr>
<td>safety radius [cm]</td>
<td>8.8</td>
</tr>
<tr>
<td>wheel distance [cm]</td>
<td>9.5</td>
</tr>
<tr>
<td>wheel radius [cm]</td>
<td>2.0</td>
</tr>
<tr>
<td>maximum speed [rad/s]</td>
<td>4.0</td>
</tr>
<tr>
<td>minimum speed [rad/s]</td>
<td>-4.0</td>
</tr>
<tr>
<td>motor noise</td>
<td>0.01</td>
</tr>
<tr>
<td>position noise</td>
<td>0.05</td>
</tr>
<tr>
<td>Orientation noise</td>
<td>0.05</td>
</tr>
<tr>
<td>IR noise</td>
<td>0.15</td>
</tr>
<tr>
<td>IR environmental noise</td>
<td>0.20</td>
</tr>
<tr>
<td>side IR spread ratio</td>
<td>0.35</td>
</tr>
<tr>
<td>IR radial position [cm]</td>
<td>6.0</td>
</tr>
<tr>
<td>IR mounting offset [cm]</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table 1. Robot and sensors sizes.

The quantities used in building the agent models are considered global variable, except for the geometrical ones. In particular the noise ratio used in several functions are tuned in such the way to give a realistic description of the real robot behaviour.

The simulation provides for several evaluations of the same genotype to figure out the fitness score. Most the times the agents are placed in the environment at each evaluation as initial conditions. In case of multirobot simulation, the starting placement is done preventing the robots to be out of range of each other perception. At the same time the robot are placed in not collided position, i.e. the initial distance is larger than the collision distance imposed as global variable.
The starting position is selected according to the empirical rule

\[
\left( d_{ij} < \left( (n-3) \frac{R}{5} + d_{\text{collision}} \right) \right) \lor \left( d_{ij} > \left[ 1 + (n-3) \frac{3}{4} \right] \right)
\]

where \( d_{ij} = \| C_j - C_i \| - 2R \) is the distance between robot \( i \) and \( j \), \( n \) is the number of robots. The position is iterated until the condition is satisfied.

III. Collision avoidance

1. Introduction: evolutionary strategy, abstraction during simulation, bootstrap problem

The multirobot context brings together potential collision during simulation and the well know bootstrap problem. Collisions happen during simulation whenever two or more agents are closer than the collision distance, set by default in the robot model. From a geometrical point of view, a collision occurs when two or more circles representing the collision radius intersect each other. The collision radius takes into account the hardware radius of the modelled robot and the safety distance to prevent crashes in the real robot behaviour (see robot model section). The collision avoidance mechanism is based on geometrical rules among circular entities, so the collision event is meant to be related intersection of circles representing physical and safety distances. The single point intersection is denoted as contact (between the radial safety areas of the agents), while the double point intersection is denoted as interpenetration. The latter phenomenon is not supposed to happen in the real robot behaviour because the NN controller should have evolved the avoiding collisions capabilities. As a result of evolution the controller should interpret signals from proximity sensors, for instance, to keep distances from other objects in the world not closer than the collision distance. This behaviour emerges from evolution without hand-coding rule as one of the first skill during the process. From experience in testing the earlier genotypes in evolution, collision avoidance capabilities are from the first to be developed. This is mainly due to strong punishment given to crashes occurred. Nonetheless the collision avoidance strategy emerges from a geometrical and virtual mechanism based upon repositioning crashed agents after collision. This means to alter the actual position of agents during the simulation with an instantaneous displacement from the collided position. In real behaviour this is clearly not possible, but the mechanism is supposed to be used only during evolution as an hint to set the relationships between inputs and outputs in case of collision.

The simulator implements also the traditional way of managing collision during evolution. Whenever a collision occurs, i.e. the circle of given collision radius intersects another geometrical object in simulated environment, the collided agent is stopped in the collided position. Since the number of crashes is punished in fitness function, the resulting agent score is strictly dropping. Moreover, the simulation stops after a given number of occurred crashes, so if the agent is kept in the collided position and not replaced, the collision occurs at each time step of simulation, increasing the number of crashes and ending the simulation. In this way the controller which lets its agent crashing has a very short life and the genotype is paid with very low fitness. This may be a problem in a multirobot context, in particular with a large number of robots populating the environment. In these cases the initial steps of each simulation run the risk of multiple and repeated collisions. This situation becomes more strict when the initial population is large, packed in a relative small area of simulated space and the starting configuration leads the agents towards the maximum agent concentration direction. The closeness of agents in initial population is due to letting the controllers perceive significant signals, namely above-noise values of proximity sensors related to other agents presence. This situation is not supposed to be set in all experiments but is relevant whenever a packed configuration occurs during simulation. The probability of falling into the bootstrap problem is higher with the increasing number of agents involved. This is mainly due to the stopping and not replacing mechanism that leads quite towards the end of simulation as soon as some collision is detected.

The principle of the proposed and implemented way of managing the collision aims at giving the agents another chance during eth simulation, replacing the crashed ones in the best escaping direction. This artefact follows the rules of evolving neural networks and doesn’t break the direct relationship between the input reading and the output setting. The replacement is done after the output action, so that the NN is not faked in its effects. If the output action drives the agent to a collision the score is punished and the agent is displaced immediately before the following inputs reading. The displacement may be more than a single step motor displacement, for this reason it
is considered virtual and not usable in real robot behaviour. After the displacement the agent perceives different inputs, now that is more far away from the previous colliding object. In this way the controller has the chance to change its response to inputs and driving another way off the crashing area. The noise contribution in setting direction and velocity at any time steps helps to escape from the colliding area. The not-surviving genotypes are those that go on falling into crashes, the surviving ones are those changing response during the simulation instead of dying abruptly. The mechanism is so based on finding the most useful way of escaping the colliding area for each agent involved in crashes. This way is found computing the resultant vector from all the distance segments between the collided agent and the obstacle object. The composition of distances and directions of segments gives the most packed area. The opposite direction is the best candidate to be the way where to escape to.

2. Replacement algorithm.

The algorithm for finding direction and displacement for replacement is computed for collided agents. So first each alive agent must be checked for crashes. In this case, the resultant vector is built considering all the objects collided with the agent. For each object (robot, line, cylinder) the Cartesian coordinates of the connecting segment are used to update the resultant vector components. Then the direction of displacement $\theta_{\text{displacement}}$ is the arctangent between the Cartesian components of the resultant vector $\mathbf{V}$ reversed by $\pi$ angle.

$$\theta_{\text{displacement}} = a \tan \left( \frac{V_y}{V_x} \right) - \pi$$

The displacement is done according to fixed frame, so that angles are absolute from x-axis. Since the group of robots and objects may be very large, the algorithm is completed by a check for other objects presence in the direction of displacement. If there’s something potentially colliding in the computed direction, the displacement is decreased until the agent doesn’t get in touch with surrounding objects. The algorithm is performed by a do-while loop for general purpose use. The displacement check may not solve collision or interpenetration between agents and objects, reducing the requested displacement for unpacking the crash. Plus, the value of displacement is generally kept very low, up to the collision radius, in order to not give a quite different readings to the controller for the following time step. More, if an agent is very close to a fixed obstacle, pushed by a large number of other agents, it may not be able to displace enough during the first loop, waiting for other agents letting room where to be displaced. Finally, the algorithm is coded for general values of interpenetration. In real robot behaviour or in simulation the displacement steps (motor and wheels action) are not large enough to let the agent overlap other entities for more than $\Delta s = v_{\max} \Delta t$.

For all these reasons the algorithm is coded with a (sometimes non efficient) do-while loop.

The algorithm structure is coded as follows:

DO\{
  FOR each agent
  IF the agent is alive..
    FOR all the surrounding objects
      Detect if a collision occurs,
      If yes, compute the distance segment
      between the agent and the obstacle
      increase the number of crashes
      and check if the agent can remain alive
    FOR all the surrounding objects
      Detect if a the object is in the displacement direction
      If yes, decrease the displacement value
  FOR each agent
    IF the agent is alive..
      Update the position of the agent
      According to displacement and direction computed
  {WHILE (at least one crash occurs)
Figure 1. Replacement algorithm after collision. 1) robots collided. 2) the vectors parallelogram is used to compute the most crowded direction, then to reverse it by $\pi$ and find the new position for robot 3. 3) similarly the vectors are found also for collided robots 1 and 2. A new collision is detected for the displaced robot 3 with the wall. The overlap is figured out to reduce the initial displacement. 4) The robot 3 is correctly replaced without collision with the wall. The robots 1 and 2 are replaced with a reduced displacement up to the value of the radius. (The circles in the figure are related to safety distance considered in the simulation.)

The first value is equivalent to

$$\vec{V}_{\text{disp}(j)} = (C'_{jf} - C_{j0}) = \sum_{i=1}^{N_r} (C_{ij} - C'_{j0}) + \sum_{i=1}^{N_s} d_{\text{overlap}}(i)$$

with $j$ robot to replace, $i = 1...N_r$ cylindrical obstacles collided by $j$ or $i = 1...N_s$ linear obstacles collided by $j$, $C'_{jf}$ first step position of resultant vector vertex for $j$ displacement (to check for further collisions) and $d_{\text{overlap}}(i)$ overlap.

The resultant vector is computed iteratively during each object detection. If the robot is collided with the object, the vertex of displacement vector is updated according to the centre of mass balance in adding contributions

$$C'_{jf}(i) = \sum_{i=1}^{j-1} C'_{j(i-1)} + C_{ij}$$

Then the computed value is checked iteratively if any further collisions are detected. For instance the collision with the wall in Figure 1 prevents the robot to be replaced at the first iteration value of displacement (regardless to the comparison with the radius value, see below). The displacement is decreased by the interpenetration.
If not necessary, the displacement for the collided agents is limited to the value of the radius in order to perturb the system as little as possible.

\[ \text{disp}_{\text{eff}} = \min\{R, \text{disp}'_{\text{eff}}\} \]

The iterations in the do-while loop provide the minimum safe displacement according to arising new collisions or too large displacements keeping the same direction computed at the first iteration. It may be possible in highly packed configurations that the displacements were larger than the radius value, or that the final configuration was quite far from the initial one or that not collided robot were involved in replacing procedure. This is shown below in some examples of replacement with simulated geometrical objects, randomly placed. The algorithm applied in simulator is rarely supposed to be used with highly interpenetrated packs, so the usual displacement are no larger than the radius value.

3. Parallelogram contributions computing.

The distance segments for the parallelogram update are computed in the same way for cylindrical objects or other robots. The collision is detected if the distance is less than twice the collision radius, i.e. the collision circles are intersected. For radial symmetry the segment is computed on the centre-centre distance. Regarding to line objects, the interpenetration value is computed according to line-circle intersection.

If an intersection occurs two points on the segment, noted as

\[ \mathcal{P} = A + r(B - A) \]

Belong to the circumference centred in C with radium R, whose equation is

\[ (x - C_x)^2 + (y - C_y)^2 = R^2 \]

Since \( \mathcal{P} = (x, y) \), and

\[ \begin{cases} 
  x = A_x + r(B_x - A_x) \\
  y = A_y + r(B_y - A_y) 
\end{cases} \]

the Cartesian coordinates system is substituted in circumference equation

\[ (A_x + r(B_x - A_x) - C_x)^2 + (A_y + r(B_y - A_y) - C_y)^2 = R^2 \]

So that the resulting second order equation is expressed for the variable \( r \)

\[ (A_x - C_x)^2 + 2r(A_x - C_x)(B_x - A_x) + r^2(B_x - A_x)^2 + (A_x - C_x)^2 = R^2 \]

the two resulting values of \( r \) are

\[ r_{i,2} = \frac{-b \pm \sqrt{4ac}}{2a} \]

where

\[ a = (B_x - A_x)^2 + (B_y - A_y)^2 \\
 b = 2[(A_x - C_x)(B_x - A_x) + (A_y - C_y)(B_y - A_y)] \\
 c = (A_x - C_x)^2 + (A_y - C_y)^2 - R^2 \]

If the value \( \Delta = 4ac \geq 0 \) an intersection occurs. The two solutions locate the points \( \mathcal{P}_1 \) and \( \mathcal{P}_2 \). If the solutions are coincident the collision occurs at tangent point \( T \).

\[ \Delta = 0 \]

\[ r_1 = r_2 \]

\[ 0 \leq r_1, r_2 \leq 1 \]

\[ \Delta > 0 \]

\[ r_1 \neq r_2 \]

\[ 0 \leq r_1 \leq 1 \]

\[ r_2 > 1 \]

---

**Figure 2. Segment-circle intersection**

**Figure 3. Segment-circle intersection cases**
As for point equation, the intersection occurs for segment if values of $r_{1,2}$ let the point $P$ to belong to finite $(B - A)$.

The vertex finding contribution for segment collision is equivalent to the overlapping value $d_{\text{overlap}} = R - \|S - C\|$ where $S$ is the mid point on the segment $(P_1 - P_2)$ perpendicular to the circle radius.

$$S = \frac{P_1 + P_2}{2}$$

4. Examples.
Some example are provided to show the replacement mechanism.

In the first example, the pack 2-3-4 is very tight so the displacements are quite large and the displacement 2 affects the initially not collided position of 5.

In the second example the resultant vector for 3 is almost null because the best choice is not to move since it is in the middle of the pack.

The last example shows a collapsed chain where the centres are very close. The replacing mechanism let the agents to restart in a similar configuration with barely changed sensors readings. The ending 1 and 5 show the largest displacement due to the iterative check for left collision.

Figure 4. replacement examples. Collided configuration (a) and solved positions (b) with centre displacement traced.

Figure 5. Usual collided configuration for 15 agents
The figure 5 shows the usual situation during multi-agent simulation where collision happen. The packs, especially groups 1 and 2, are separated into the un-collided similar configuration by short displacements. The figure 6 shows an highly packed situation solved with larger displacements.

IV. Proximity Sensors

1. Introduction.
The interaction of the robots with object in the simulated world is based on inputs from proximity sensors. Since the world is modelled by polygonal objects, the perception of objects populating the world must be based on geometrical rules for fast and reliable simulation.
The use of look up table recorded with samplings of sensors may be referred only to objects used during the sampling. The proximity value recorded by sensors of a robot facing another robot are quite different from values recorded facing the wall. This is due to the irradiation shape of IR emitters and the way the receivers get waves from many directions. So each sensor is affected by what happens in the nearby emission and reflection of waves. Since is impossible to access to a sampling database for each type of object faced during simulation, an ideally geometrical model for sensors behaviour is implemented to give account of inputs. The model should rely on directions and angles to interact with other geometrical models. But this means to give up with the real model of IR signal. However, the main purpose of the simulator is to provide the environment for evolving the neural controller, and it must be fast and flexible. The only constraint is to guarantee the neural controller to be provided with reliable and realistic inputs, so that once ported into the microcontroller of the real robot, it performs the right behaviour and outputs driving.

As the IR signal is quite noisy, and the sensor on the modelled robot as well, is not worthy to figure out a 2D or 3D model of IR bulb emitted and received. The model provides on the contrary a first directional information of the distance between the emitter and the object to detect. This
is the basis on which the noise phenomena are modelled and added. This method has been proved to be enough reliable by the porting test on real robots. As a result of the statistical analysis of noise recorded in IR sampling during the real behaviour, the distribution of noise added to initial signal value from geometrical model is simulated in a way very similar to the real one. Plus the neural controller is trained to face the worst condition in the simulated world from a signal reliability point of view.

The final values provided to the inputs are so computed from signal idealization and noise distribution. The basic signal model is computed from several records of sensors sampling on the robot facing the wall. Then the values are processed to give a function for fast simulated signal response from geometrical variables.

2. Sampling

The target model for IR signal is geometrically provided by a function like \( IR_{reading} = f(d_{samp}, \theta) \). The sampling phase is set for record IR values from different distances and directions. The initial set up is shown in figure 8.

![Figure 8. IR sampling set up.](image)

The robot is placed to fixed distance from the wall. The first available distance \( d_0 \) is the one that let the gripper in front side to brush the wall. The first position means contact with objects both in real and simulated world. For each \( d_i \) position the robot is completely rotated letting the whole set of sensor to face the wall. The cycle is divided into \( m \) position. For each position \( S \) samples are collected for each sensor. Then the distance is increased until \( d_f \) where the values are detected as noise in every direction. A post sampling evaluation give the value of sensors range as the last distance to which a not noisy value can be detected.

<table>
<thead>
<tr>
<th>( N )</th>
<th>Number of IR sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_0 )</td>
<td>Initial distance [cm]</td>
</tr>
<tr>
<td>( d_f )</td>
<td>Final distance [cm]</td>
</tr>
<tr>
<td>( \Delta d )</td>
<td>Displacement step [cm]</td>
</tr>
<tr>
<td>( m )</td>
<td>Number of positions</td>
</tr>
<tr>
<td>( \theta_0 )</td>
<td>Initial position [°]</td>
</tr>
<tr>
<td>( \theta_f )</td>
<td>Final position [°]</td>
</tr>
<tr>
<td>( \Delta \theta )</td>
<td>Rotation step [°]</td>
</tr>
<tr>
<td>( S )</td>
<td>Number of samples</td>
</tr>
</tbody>
</table>

Table 2. Sampling set up.

The resulting raw data tables are processed for averaging the \( S \) samples and to find the relationship between direction, distance and values.

![Figure 9. Records tables.](image)

Since the sensor mounting on the robot border is different from the rotation step during the sampling procedure, the recorded values are referred to uniformly distributed angles for each distance placement. Moreover the effective distance between each sensors and the wall in each sampling position depends on facing angles. For this reason the whole set of collected data so clustered into distance and angle intervals according to geometrical set up shown in figure 10.
For each rotational position, every sensor faces the wall according to distance and angle depending on rotation of the robot and relative position on the robot turret.

\[ \phi(m,n) = \theta_0 + (m-1)\Delta \theta + \theta_{\text{turret}}(n), \text{ with } \phi \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \]

The more useful angle expression for simulated and computed geometry is

\[ \phi(m,n) = \psi(m,n) + \frac{\pi}{2}, \text{ with } \psi \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right] \]

where \( m \) is the rotation step, \( n \) the sensor.

The resulting distance between the robot border and the wall under the angle computed is

\[ \text{read}(m,n,samp) = \frac{d_{\text{samp}} - R \sin \psi(m,n)}{\sin \psi(m,n)} \]

In this way an overall table with 3-element arrays is built. The elements are effective distance \( \text{read} \), angle \( \psi \) and the sampled value. The differences and noise during signal acquisition are share among all sensors in the equipment set. The signal model is figured out as unique geometrical result from all the available records.

3. Interpolation.

The clustering process gives the interpolated surface \( IR\text{reading} = f(d, \theta) \) from samples.
The analysis of variance in averaging the values along the angle axis for each distance values gives a value comparable to overall noise. 
\[ \sigma \in [0.0813, 1.6757] \] and since the maximum recorded value for IR signal is approximately higher than 40 in sampled scale the order of magnitude of standard deviation is approximately up to 5%. Since the basic reading value must be affected by noise is worthy to simplify the reading model giving up the direction information and speeding up the computing of reading value from the distance. The noise contribution added to basic value provides a realistic input for the controller, as proved in porting test.

The resulting averaging process reduce the dimension of function domain and let the sampled values to be interpolated by a polynomial function. The interpolation technique is simply the least squares regression.

The resulting function is

\[ \text{read}_{\text{int, exp}}(d) = 35.7606 - 16.0115d^2 + 6.1513d^3 - 0.8072d^4 + 0.0261d^5 - 0.0026d^6 + 0.0001d^7 + o(d^8) \]

The choice of the order of interpolating polynomial is due to the effect on near field values. At closest distance there are no available values in sampled tables because of the gripper offset in initial position. But placing the robot close to the wall without rotating the turret, the sampled values are higher than 30. The chosen polynomial is the best among the low order ones in giving account of the closeness readings. The polynomial is not computed during simulation at each time step for each sensor because of computational slowing down. The function is tabled and the look up with the computed distance provides the reading value. The range value is fixed to 15 cm, after which samples are considered noise.

The shape of the function is slightly different from the exponential model proposed in another simulator for the same robot. This new one is kept because of the absence of statistical comparison between the two functions.

![Figure 12. Polynomial interpolation.](image-url)
The actual shape explains in satisfactory way the signal reading dropping at the distance of approximately 5 cm from the turret border. This is observed also in real robot behaviour and is probably one of the reasons of keeping the dropping threshold as stable distance among other robots during motion.

As the linear beams are traced around the robot, the ideal signal is so detectable as intersection of the traced segments with objects in the world. The intersection value gives the distance according to which the reading is computed. The IR sensor emitted signal is so replaced by a line beam, reducing the detection of object to the narrowness of the beam. In some cases, as shown in figure 7 world, the simulated narrow beams cannot intersect objects laying in the perception range even if the real shape of emitted signal gives back some readings. As a result a tight discontinuity in sensor readings is presented to the controller inputs. In these cases the side readings are somehow spread to the geometrically blind sensors. If a beam doesn’t intersect anything but the adjacent beam gives valid reading, a percentage of the reading is assigned to the blind one. The tested distribution in spreading the readings suggest to keep an approximately one-third value.

The noise in reading is given as uniformly distributed variable in the 15-20% of reading for each sensor. The average noise in porting tests suggests a 10-15% of noise in samples recorded. Anyway the simulator is implemented with more strict condition to (contribute to) improve the robustness of the inputs handling by the neural controller.

As a last result of the porting test a persisting and all around the robot distributed noise is noticed. This kind of environmental noise is shown in all the sensor readings, in the objects facing sensors as well as the free space ones. The scaled values are usually included between 0-4, so an uniformly distributed value from this range is finally added to the basic noisy reading.


5. IR sensor distance computing with segments.
   The distance used in the look up table for IR readings is traced as the portion of beam segment (IR range of 15 cm long) intersected with cylindrical or linear objects.
   For linear beams the two segments intersection rule is used. The intersection point is common for both the segment representing the IR beam and the segment representing the object.
Figure 14. Segments intersection.

\[ P = A + r_1 (B - A) = C + r_2 (D - C) \]
The planar equation is expressed for the variable \( r_1 \) and \( r_2 \)

\[ \begin{align*}
A_x + r_1 (B_x - A_x) &= C_x + r_2 (D_x - C_x) \\
A_y + r_1 (B_y - A_y) &= C_y + r_2 (D_y - C_y)
\end{align*} \]

Substituting \( r_1 \) in the second equation, \( r_2 \) results

\[ r_2 = \frac{(C_y - A_y)(D_x - C_x) + (D_y - C_y)(A_x - C_x)}{(B_y - A_y)(D_x - C_x) - (D_y - C_y)(B_x - A_x)} \]

\[ r_1 = \frac{(A_x - C_x)(D_x - C_x) + (B_x - A_x)(D_x - C_x)}{D_x - C_x} r_2 \]

The intersection is real if both \( r_1, r_2 \in [0, 1] \).

Since the segments are finite, values \((r_1, r_2 > 1) \lor (r_1, r_2 < 0)\) are related to intersection point out of the ending vertexes. The value of distance used for computing the proximity reading is \( r_1 (B - A) \), or the fraction \( r_1 \) of the valid range.

<table>
<thead>
<tr>
<th>IR</th>
<th>angle[°]</th>
<th>( r_1 )</th>
<th>( r_2 )</th>
<th>read?</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>34</td>
<td>0.34</td>
<td>0.60</td>
<td>v</td>
<td>5.220</td>
</tr>
<tr>
<td>1</td>
<td>56</td>
<td>0.40</td>
<td>0.70</td>
<td>v</td>
<td>6.087</td>
</tr>
<tr>
<td>2</td>
<td>79</td>
<td>0.60</td>
<td>0.88</td>
<td>v</td>
<td>9.07</td>
</tr>
<tr>
<td>3</td>
<td>101</td>
<td>1.58</td>
<td>1.56</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>120</td>
<td>-3.00</td>
<td>-1.38</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>140</td>
<td>-0.76</td>
<td>0.11</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>160</td>
<td>-0.47</td>
<td>0.35</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>180</td>
<td>-0.37</td>
<td>0.46</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>200</td>
<td>-0.34</td>
<td>0.54</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>220</td>
<td>-0.35</td>
<td>0.62</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>240</td>
<td>-0.42</td>
<td>0.72</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>259</td>
<td>-0.60</td>
<td>0.88</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>281</td>
<td>-1.58</td>
<td>1.56</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>304</td>
<td>1.87</td>
<td>-0.63</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>326</td>
<td>0.63</td>
<td>0.21</td>
<td>v</td>
<td>9.542</td>
</tr>
</tbody>
</table>

Figure 15. An example of beam intersection with segment (right) and corresponding values of intersections and distances (left). Angular position is related to real position on robot turret.

6. IR sensor distance computing with cylindrical objects.

The effective distance between the sensors and the circular object is computed only for those sensors facing the object. First the perception is detected, i.e. the presence of the object in the sensor range field.

\[ rHdist < \left[ \text{radial position(sensor)} + \text{range(sensor)} + R \right] \]

Then the geometrical distance is used to compute the absolute direction of the centres connecting segment. \( \theta_{abs} = a \tan \left( \frac{rHdist_y}{rHdist_x} \right) \)

According to relative position, the relative angle of the sensors mounted on the robot turret must be referred to fixed frame.

\[ \theta_{facing} = \theta_{sensor} - \theta_{rel} \quad \text{with} \quad \theta_{rel} = \theta_{abs} - \theta_{orient} \]

If the angle corresponding to each sensor position belong to the facing interval, the distance value is computed. The angular range
depends on the distance between sensors and the object and relative size of the latter. The closest the object, the wider the facing angle.

\[
\max\{\theta_{\text{facing}}\} = a \sin \left( \frac{R}{rH_{\text{dist}}} \right)
\]

The effective distance slightly depends on the facing angle for circular shape. It is equivalent to

\[
dist = rH_{\text{dist}} \cos(\theta_{\text{facing}}) - R \left\lceil a \cos \left( \frac{rH_{\text{dist}} \sin(\theta_{\text{facing}})}{R} \right) \right\rceil - R
\]

Figure 16. Effective distance computing with cylindrical object.