Flock Distortion:

A New Approach In Mapping Environmental Variables In Deep Water

David McFarland, Ehsan Honary
Intelligent Autonomous Systems (Engineering) Laboratory
University of West of England

Abstract

This paper demonstrates a new method in sampling environmental variables and underwater currents using special devices with a particular technique. The project involves developing a set of robots as the tools for data collection, and developing a system that can interpret the data collected. The 'ocean' currents can be mapped using a flock of robots by dropping them into the ocean. As they descend, they record the desired sensory information such as salinity, temperature, pressure, the presence of specific chemicals, etc., along with other variables required for the system to function. Once at a predefined depth the robots start ascending and still continue collecting data until they reach the surface. Back on the surface, they communicate the collected data to a base computer in a ship or on land, through satellite or other means. The base computer collects the data from every recovered member of the flock and reconstructs the path they took underwater. Then using this path, it estimates the vectors of the oceanic currents. The environmental variables can be displayed in 3D for the sampled area.

1. Introduction

Detecting the underwater oceanic currents has long been a dream of meteorologists and oceanic scientists. Such information would help to explain some of the questions about oceans, weather patterns, origin of life, the creation of earth and so on. However the task has remained a tough challenge. This is due to large expanse of the ocean and our limited ability to monitor it. Over the years different instruments have been developed for sampling the most important environmental variables (salinity, density and temperature) but each has its own limitations.

A type of platform used commonly in modern oceanography is the *anchored buoy system*[1]. Surface sensors can be attached to them, used for weather conditions. Subsurface sensors can measure currents, temperature, or other parameters. The collected data can then be transmitted via a satellite.

Another method is to use *popup profiler,* developed by WHOI, Woods Hole Oceanography Institute. This tripod device is lowered to the sea floor, where it releases acoustic beacons. The beacons slowly rise to the surface. The position of the beacons is tracked by the profiler, which is at the bottom of the sea. Eventually the profiler is recalled electronically from the seafloor to the surface and the data are retrieved. The currents can be calculated using the affected trajectories.

An important method commonly used to sample ocean currents is using Acoustic Doppler Current Profiler (ADCP). This device uses sonar and the reflection on the particles in the sea to detect the relative velocity of the current. If the velocity of the ADCP device is also known, it is possible to measure the velocity of currents. The benefit of this device is that it can measure the current from far away without disturbing the current itself. ADCP can be mounted in a fixed location. However, if ADCP is deployed on a moving vessel, as is commonly used in ocean sampling, the velocity of ADCP should be known at all times. In practice a surface vessel is used to track the device. This is costly and is difficult to deploy everywhere (such as under icecaps).

In recent years another instrument has been used extensively for this purpose. It is called *ALACE, Autonomous Lagrangian Circulation Explorer* developed by WHOI [2] . This is an autonomous subsurface float which has a fixed behaviour cycle. The cycle starts at the surface. The float then submerges to a neutrally buoyant depth and is carried away by the currents in that depth. It samples the temperature and salinity of water during this time. The float then ascends to surface where it transmits the collected data to *System Argos Satellites*. The cycle continues again[3]. A common variety of this is S-PALACE, Salinity Profiling ALACE.

There are a few problems with this method. The float has to be on surface for 24 hours for data communication. This is wasteful for batteries, but perhaps the most important of all is the lack of knowledge of the trajectory of the float when it is underwater. The only known geographical parameters are the start and final positions in a particular cycle, and hence it is assumed that the float travels straight from start to end in a cycle, which is very unlikely.

Argo is a project for observing the oceans globally. It uses a global array of autonomous profiling floats[7]. The resolution of this system is around 200-300 km, which is useful for climate and oceanography scientists. The system consists of floats that are similar to PALACE probes described above. They are usually dropped into the ocean from a surface ship. After deployment, they descend in water up to a predefined depth (around 2000m). They are then drifted by the currents. They sample the ocean for temperature and salinity during the drift. After a certain time, they ascend and transmit their data through a satellite to some land base. The collected data is then calibrated and evaluated before it is provided to scientific community. This cycle is repeated for each float. The floats operate around 3-4 years. Each cycle is approximately 14 days. The floats remain almost a day on the surface for transmission of data. In general communications uses about 50% of a float's energy budget.

The movement of the floats is graphed using straight lines for each cycle of the floats. The interconnected lines show the movements of the floats caused by the underwater currents. The system would eventually have 3300 floats each profiling around 25 times every year and the data is almost real-time (1-2 days delay).

Argo is a good system for understanding and observing the oceans in a truly global way. However, the resolution of the sampling is not high with this system. For high-resolution applications, there is a need for another system.

The methods described so far, simply are not adequate to provide a reasonable way of autonomously mapping environmental variables in the deep ocean with high resolution, even if the memory and processing power is provided.

An example for ongoing projects, is the aim of the team behind *Neptune*[4] which monitors a particular area of the ocean permanently. The *Neptune* project is about making an underwater communication LAN in the Juan De Fuca plate near Seattle and link it to the Internet so the data collected from the ocean could be continuously logged and observed for a long time. This is a big project, which requires an array of sensors and communication bridges to be built on the bed of the ocean. Underwater autonomous vehicles would then explore and collect data, including the direction of the underwater currents, and can transfer the collected data when they are close to a docking station or when they actually dock for recharging. This method is effective for the area under observation, but it is not practical to make such an array each time a particular area of the ocean needs to be observed.

2. A new method

Our suggested solution to the above problem is to detect the currents through the deformation of a flock of probes, or robots. The basic idea is that when a flock of objects are thrown into the ocean, the shape of the flock is distorted as the flock experiences underwater currents. If this deformation can be detected and recorded, it should then be possible to work out the speed and the direction of currents using this data. Each robot is capable of recording desired environmental values such as temperature, pressure, salinity, chemical pollutants, etc., along with the rest of the information needed to detect the flock distortion.

The idea is that a flock of robots would be dropped into the ocean from a ship or an airplane. The robots hit the water at a particular location. They will be constructed in such a way to float for a few seconds before they start to descend. In this time they can record their location using GPS. This information is saved in each robot. The robots would then descend with gravity. All the robots will be synchronized in time, and at each time step they can record the input from their sensors. They will record the depth, the salinity of water, the water temperature, and so on. In addition, each robot will record the distance between itself and all other robots. This will probably be done by using sonar with different codes so the robots can have separate identities. However, it is not necessary to monitor the bearing of other robots. This makes the robots a lot simpler and cheaper to make. Robots designed for this role, are currently being developed[5] .

As each robot descends it records all the environmental information, and the distances to the other robots until it reaches a certain depth. At this depth it stops descending and starts ascending to the surface. On the way up the robots still continue to record the information. It is likely they will be coming up in a different area of the ocean, so the data recorded during the ascent is just as useful as that taken during the descent.

When they reach the surface of the water, the robots locate themselves using GPS and then transmit their position along with the rest of the collected data to a computer in a particular land base via a satellite. The robots can be collected from the surface of the water, or they can be abandoned if collection is considered too expensive. The whole sequence is illustrated schematically in Figure 1**.**

The base computer collects all the data from the robots that managed to surface and transmit their data. Then the computer runs an algorithm, called the *flock estimation algorithm*, to estimate the absolute position of the flock as it descended, and tries to reconstruct the path of the robots through water. The velocity profile of the currents that the robots experienced are also calculated and this is used to reconstruct the oceanic currents. The sensory information

that has been collected at each time step can now be interpreted since the position of the robots is known when the data was collected. This means a 3D graph of a particular environmental variable can now be drawn for the sampled area.

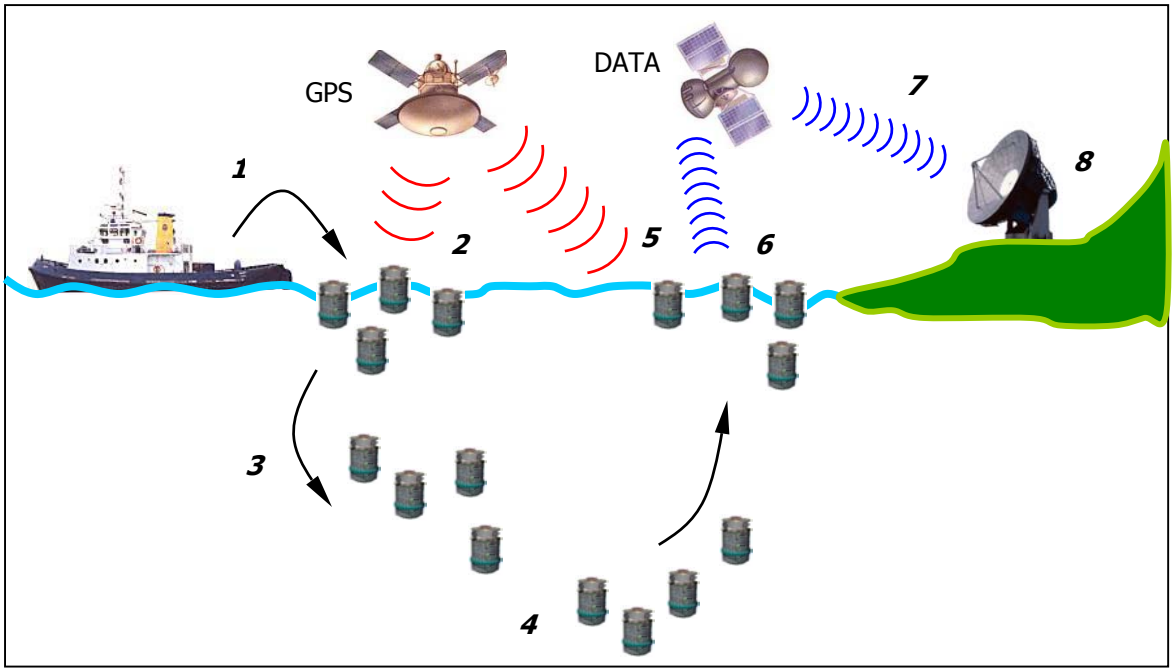


Figure 1. The components of the system. 1: The robots are deployed into the ocean. 2: The robots record their GPS location. 3: The currents underwater affect the robots. 4: At a predefined depth, the robots start ascending. 5: Robots record their final GPS location. 6: Robots send the collected data to a satellite. 7: The satellite relays the data to a land base. 8: In the base, data is collected, stored and is processed.

3. The robots

In order for this system to work the robots should be capable of doing a number of things. They should be able to:

1. Descend in a controlled manner.
2. Ascend after they have reached the required depth.
3. Function at great depths.
4. Monitor local environmental variables, such as temperature, salinity, specific chemicals, etc.
5. Monitor the distances of neighbouring robots.
6. Log all the information they monitor.
7. Transmit the logged information to a suitable destination, once they have resurfaced.

In the IAS laboratory we are developing a diving robot that will meet these criteria[5]. Buoyancy is controlled by altering the density of the robot, so that the robot can head for the surface when it reaches a preset depth. In due course, the robots will carry the necessary instrumentation for flock distortion missions, and will be able to log all the data, and relay the data upon resurfacing. These robots are small and inexpensive (see Figure 2), and could be considered to be disposable. There are probably many

Figure 2. An image of the prototype robot. (height = 38cm)

robot designs that could fulfil the criteria for flock distortion missions. However, it is desirable that the robots be inexpensive, even disposable, and that the flock-distortion algorithms allow for some robot losses, and/or failure to recover data from some of the robots. Our current research has this objective in hand.

4. Data processing

All data processing is done by the base computer. Some data processing could be done on-board individual robots, but this would make them unnecessarily complicated and expensive. The base computer can be as powerful as needed. Moreover once the data is collected it could be used several times.

The tasks that the base computer has to perform are as follows:

1. Collect data from the robots that reach the surface of the ocean and are recovered, or transmit their data via a satellite link.
2. Organise the data so that it is filed as a function of time, in which the information provided by the different robots is synchronized in time.
3. Enhance these data by appropriate noise-reduction and scaling algorithms (the *distance table*).
4. Use the *distance table* to estimate the absolute positions of the robots as a function of time.
5. Calculate the velocity profiles of the robots.
6. Reconstruct the currents responsible for these profiles.
7. Map the currents onto an ocean map.
8. Show the environmental variables visually.

The major part of this program is the *flock estimation algorithm*. The input to this algorithm is the *distance table*, which contains all the necessary information for the subsequent calculations.

The *flock estimation algorithm* has three main parts:

1. *Relative estimation algorithm.* This uses the *distance table* to calculate the relative path of all robots using two robots as a reference.
2. *2D/3D velocity profile estimation algorithm.* Using the results of the previous algorithm, the velocity profile and the absolute path of *seed robots* are calculated.
3. *Absolute estimation algorithm*. Using the results for *seed robots,* the absolute path and velocity profile of the all other robots are calculated. This also applies any enhancements required and checks for the validity of the results.

A block diagram of this program is shown in Figure 3.

In real life, the *distance table* would be generated from the data collected from the robots. In this paper, we consider data collected from land-based robots (section 5), and data generated by computer simulation of ocean currents (sections 6,8 and 9).

5. Experiments with real robots in 2D

As a demonstration of flock distortion in its simplest form, we set up an experiment using mobile wheeled robots. For this, a set of UBOTs developed in the IAS lab for another

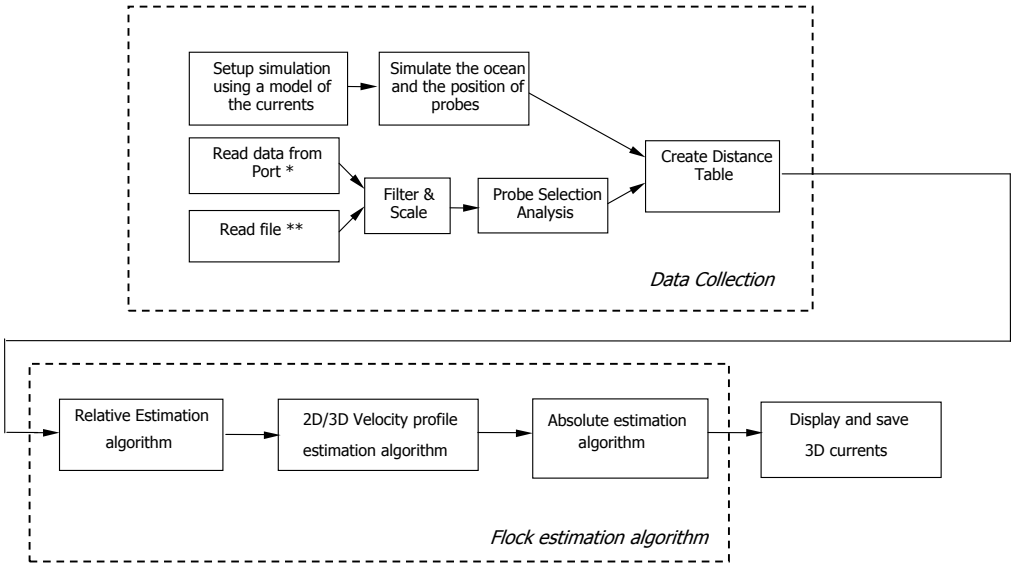


Figure 3. The block diagram of the program. (*) Data can be read from a communication port which has the data collected from the robots. (**) The data can be read from a previously saved file.

purpose were used (for details of these robots see Melhuish[6] and Figure 4). The robots were deployed in an arena, an octagon with a width of 9m, with a wide stripe of width 1m painted onto the floor. To start the experiment, the robots are lined up orthogonal to the white stripe, as shown in Figure 5. The robots were switched on one at a time, in sequence (representing robots sequentially dropped into the ocean from a moving boat), and were programmed to move straight ahead at a uniform speed until they reached the white stripe. They could detect the proximity of the stripe by means of a light sensor mounted under the robot**.** As each robot detected the stripe, it turned through an angle of approximately 45 degrees. (The unmarked floor of the arena represents still water, and the white stripe represents moving water. So the turn made by the robots represents the

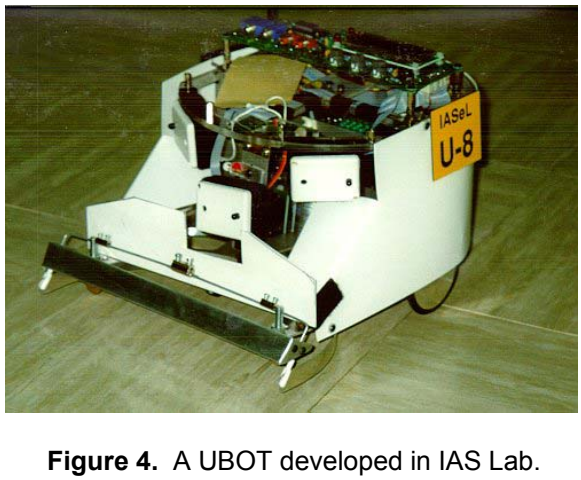


Figure 4. A UBOT developed in IAS Lab.

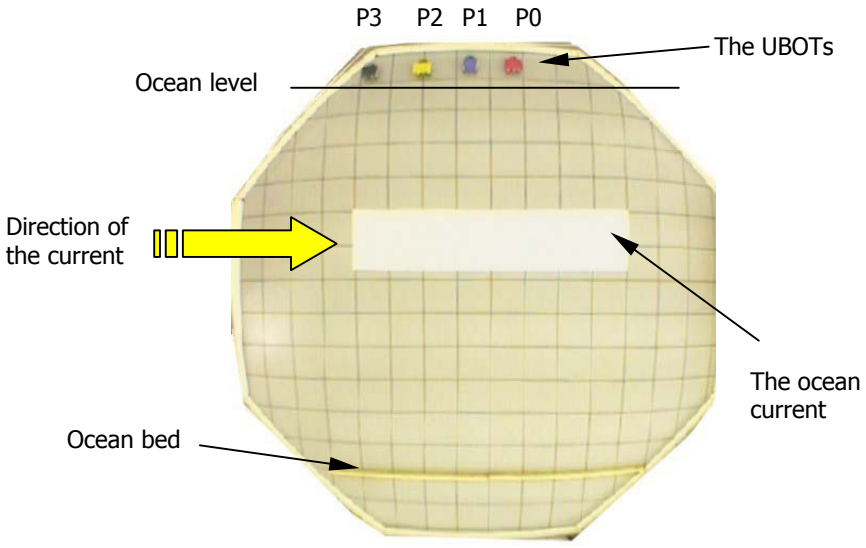


Figure 5. The UBOTs in position prior to entering the “ocean”. Top view of the arena recorded by the overhead camera. Probes P0, P1, P2 and P3 are always deployed in that order.

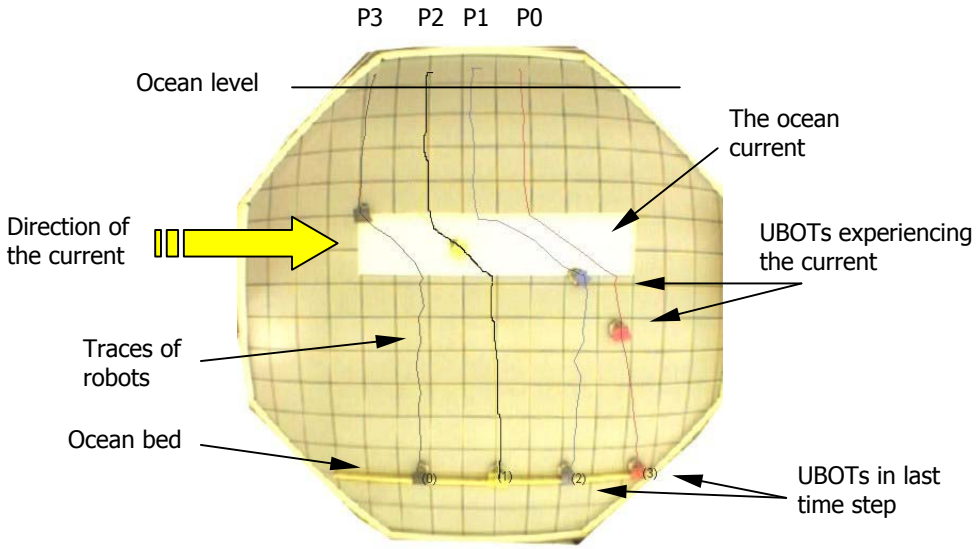


Figure 6. UBOT experiment. The white stripe is the current and the four robots have been dropped into the ocean from the top. The UBOTs are shown in two different time steps to show their movements. (View as in Figure 5).

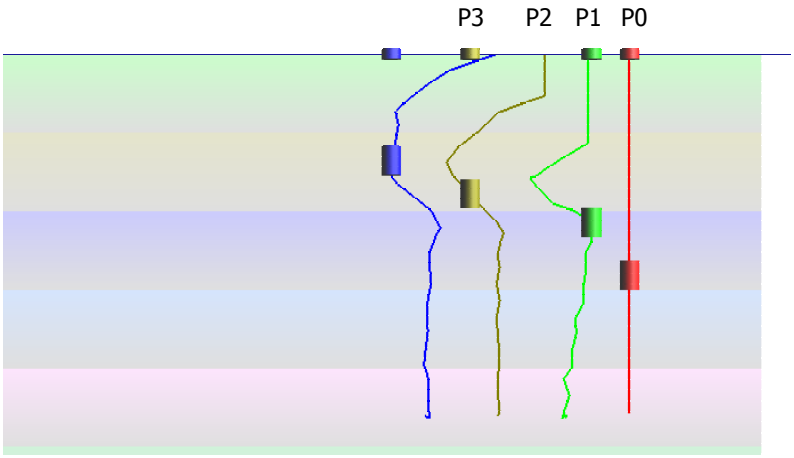


Figure 7. The results of *relative estimation algorithm* using the distance table generated by the tracking program. P0 to P3 are the robots and their traces through the "water".

movement of a passive robot caused by a current). Once the robots can no longer detect the stripe, they turn back to their original straight-ahead direction.

The robots were tracked with an overhead camera, and their positions recorded as a function of time. The trace of their path as shown in Figure 6, shows the output of the tracking system. At each frame (time unit), the distances between each robot and other robots are recorded along with the original formation parameters. From this data a *distance table* is compiled. These data are then fed into the *flock estimation algorithm* to calculate the *velocity profiles* . The result for the first part*, relative estimation algorithm* is shown in Figure 7.

The data are affected by inaccuracy in the tracking system, and noise from the robots movement, which is relatively uncontrolled. As a result the data are noisy, but nevertheless they roughly show the current. This information is used to calculate the velocity profile and the absolute values of the robots descending in the ocean using the *2D velocity profile estimation algorithm* and *absolute estimation algorithm*. The result is shown in Figure 8.

The calculated velocity profile along with the expected velocity profile is shown in Figure 9 with a higher resolution to show the details.

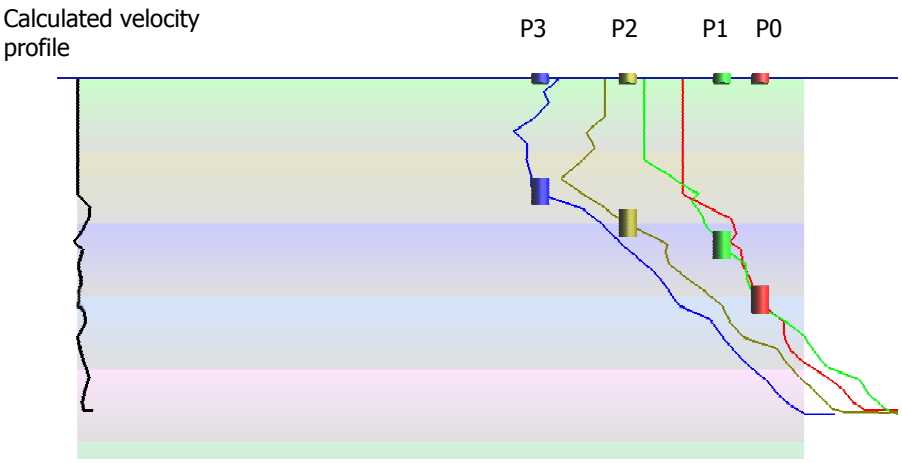


Figure 8. Velocity profile and the absolute positions of the UBOTs calculated by *2D velocity profile estimation algorithm* and *absolute estimation algorithm.* (Notation as in Figure 7).

The expected and calculated velocity profiles have some features in common, but they have some differences. These are directly due to the accumulation of error. If by any chance there is an error introduced in the traces, the velocity profile directly reflects that. The calculation is based on the requirements which indicates that the *seed robots* (here robot P0 and P1) should have experienced the same current. However in this example each robot is experiencing different currents due to errors in their movement (such as skidding or different amount of friction experienced) and also the error introduced by the tracking program when it is searching the images to find the robots. The calculation between the traces and the velocity profile is one to one. That is, no matter what the traces are, the velocity profile is calculated "as if" the seed robots have experienced the same current. This means that if the traces are a few percent out, the calculated velocity profile might have more error.

This is why the absolute values shown are drifting to the right. The accumulation of error creates a velocity profile that shows the existence of some current in a deeper section of the 'ocean' when there is none. The error is reflected in the path of the robots.

To improve the results it is possible to use multiple seeds. This means that the program tries to find the best pair that experienced the same current or just uses all possible pairs to calculate the average of velocity profiles to cancel out some of the error. The velocity profile shown in Figure 9 is actually calculated this way using three permutations.

A better way to improve the results is to analyze the velocity profile and perform some signal processing to reduce the effect of the accumulated error. This is the subject of current research.

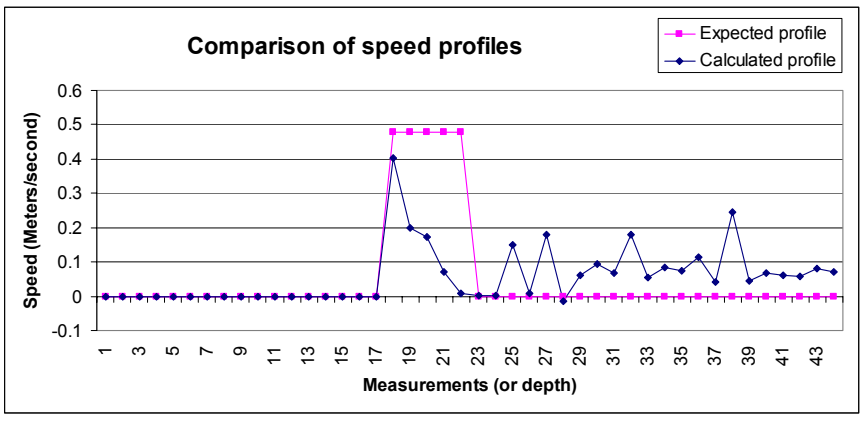


Figure 9. Calculated and expected velocity profiles. "Measurements" axis is equivalent to depth.

Finally, the principle of flock distortion is demonstrated by this experiment. The results are not as clean as we would like, but they have revealed some problems which will need to be addressed in analysing the data from aquatic robots. To this extent, this exercise has been useful.

6. Simulation of ocean currents

To demonstrate our approach, we start with a simulated environment. The ocean and the currents are therefore simulated in the program. The robots are dropped and the relative distances are then recorded. The *distance table* is then generated, so it can be fed into the *estimation algorithm* to calculate the *velocity profile* of the currents. Since the currents are simulated, they can then be compared with the estimation to judge the efficiency of the algorithm.

Different classes of currents
For the purpose of this research the currents that the simulation creates are divided into a number of classes. In this way the performance of the algorithm can be detected more easily.

Class 1: Constant horizontal currents. These currents are horizontal. At a certain depth in the ocean there is a particular value for the speed of the current and this is constant for the whole ocean. The currents do not change over time, and there are no vertical currents.

Class 2: Varied horizontal currents. These currents are the same as Class 1 except that they can be different at different positions in the ocean. This means that robots dropped in different positions in the sea might experience different currents.

Class 3: Varied horizontal currents with varied vertical element. These are the same as Class 2 and in addition they have vertical elements added to the currents. This means that at any (x,y,z) position in the ocean there can be a (V_x,V_y,V_z) vector that represents the current.

Class 4: Varied horizontal currents with varied vertical element that can change over time. As the title suggests, the velocity vector can have different values at different times. This, if simulated properly, is the same as real world currents.

In this paper we are mainly concerned with Class 1 currents. The algorithms discussed here are also capable of handling Class 2 currents. These will be discussed in a future paper.

To start with, the simulation is made to use simple models of the currents for Class 1 currents. The *velocity profile* can be defined which then will affect the movement of the probes, as they descend. The descent is simulated with a constant speed, representing the terminal velocity. The probes start outside of the water, then enter the water and go down. To keep the algorithm simple, the robots would reach the bottom of the sea and would stop rather than coming up again (This will be modified in future work). The simulation can be done in 2D or 3D. The 2D case means that the robots can only move in a vertical plane going sideways and down. Both cases are interesting since they both have applications of their own. For example a simulation in 2D and 3D is shown in Figure 10. The picture on left shows a basic current, which starts at a certain depth and finishes at a lower depth and has constant speed in between. The picture on the right has a velocity profile of a sine function in x and z direction, which is more similar to currents found in ocean. These velocity profiles

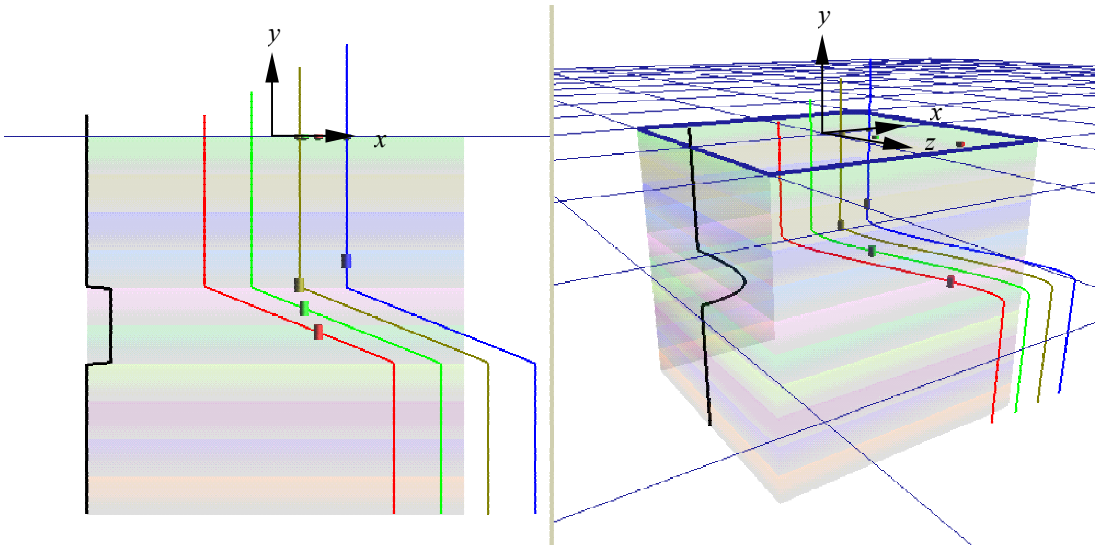


Figure 10. The simulation of the ocean. Left: Linear current in 2D. Right: Sine wave current in 3D.

are not similar to natural currents, but are good enough to test the performance of the algorithm as it is developed. The simulation also does not consider the collision of the probes, nor are there any random drifts.

Distance table

The *distance table* is the end product of the first phase of the calculations, which can be created from real data or from the simulation. It is basically the table of distances of all robots from each other at all time steps. Each individual robot has a set of data representing the distance between that robot and all other robots. This information has to be cross checked with data from other robots to see if it is consistent. It is possible that a particular slot will be empty due to the fact that the robots might not see each other because they are out of range. It is also possible to miss a robot altogether and so the information about that robot might become incomplete. These problems and the possible solutions to them are addressed in the final discussion.

7. Estimating relative positions of the robots

Once the *distance table* is created, it is passed to the *relative estimation algorithm*. The object of this routine is to obtain the relative positions of the robots at all time steps. Since the only available data are distances between robots and there is no information on bearing, this task is more complicated than it seems at first sight. The idea is to use extensive amount of geometry, triangulation and floating point calculations to estimate the position of the robots.

The information needed for this part of the program comprises the distances between robots, the original location of the robots when they hit the water in relation to world coordinates, and the depth of each robot at each time step.

One robot. A single robot would not be sufficient since there is no way to tell how it moved about when it is underwater.

Two robots. A couple of robots would have a single value for the distance and two more values for their depth. A robot can be selected as a reference robot and we assume that it goes down straight. So now the position of the other robot would be relative to the reference robot. From the reference robot's point of view, the other robot can be in a circle which is the intersection of the depth plane of the robot and the sphere with radius of distance centered at

reference robot as shown in Figure 11 (Left), but there are infinite number of solutions. So mathematically it is impossible to calculate the relative positions. If we convert this to a 2D environment, then we have only two choices to select from, Figure 11 (Right). To select between the two solutions we can employ a heuristic approach. We can use the original location of the robots, and assume a continuous movement for the robot. Therefore at each time step a robot's location should be close to its previous location. Since the two possible solutions are mostly away from each other, the right solution can be selected at each time step. However, there is a problem when the two robots are close together. In that case, the two solutions will be too close to each other to distinguish between them. Also if one of the robots crosses to the other side of the reference robot, there is no way to tell that this has happened since the distance value is always positive. To break the symmetry we need another robot.

Three robots. With 3 robots, we have a lot more information to use. If three robots are used in a 2D environment, using the third robot's distance to others it is possible to locate the position of all robots. In this case there is no need for a heuristic. The third robot's position is used to select between the two possible solutions. If they are in a 3D environment, they will form a triangle, or a line in a special case which will be the same as in 2D. The triangle can be solved easily since all the sides are known. See Figure 12. Since the depth of each robot is known, the horizontal distance between the robots can be calculated. This is basically the triangle between the robots when the robots are viewed directly from the top.

However any triangle with this shape rotated around the reference robot is also a possible answer, apart from the fact that it can also be flipped. This is shown in Figure 13. The flipping at each time step can be solved by using the heuristic and decide between the two possible solutions. But we still have the rotational problem.

Four robots and more. But what if we use more than 3 robots. Is this going to solve the problem with infinite number of solutions? Suppose we use 4 robots as shown in Figure 14. Each robot can be solved using a triangle with the same technique as described. However the extra information about the distance between the fourth robot and others is not going to help a lot. The solution can still be flipped, or rotated while staying with the same shape.

By using more than 3 robots, we can not get rid of the heuristic nor can we solve the rotational problem. The heuristic does not solve the rotational problem, since the flock can spiral while it is going downwards and the previous position of each robot does not lead to its correct position selection in the next time step. Of course there are many other benefits in having more robots as it will be discussed later, but we need another solution than keep adding robots. As it turns out the problem can be solved once the relative positions of the robots are known.

Relative estimation algorithm

Considering the above requirements this algorithm tries to get the relative positions of the robots assuming that one of the robots (reference robot) is going down in a straight line path. Also the position of the next robot is assumed to be in a particular direction, so the solution will be found for a particular rotation around the reference robot. For both of these, the original location of the robots is used as the initial values. The reference robot is always in a constant location (its original location) despite the fact that the real robot can move. The second robot (direction robot) always stays in the same direction as its original direction, however the distance between the reference robot and the direction robot can change

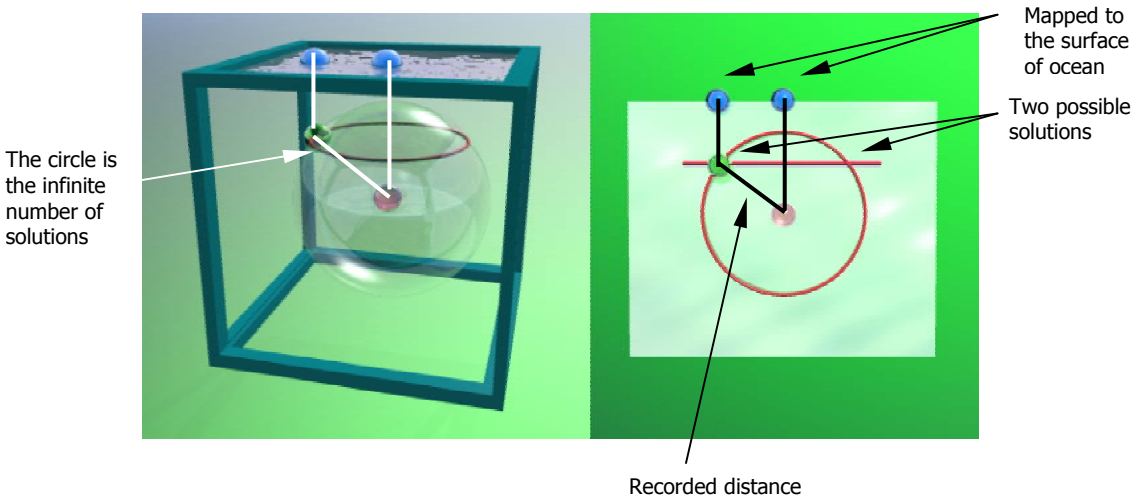


Figure 11. Two robots scenario. Left: 3D environment. Right: 2D environment.

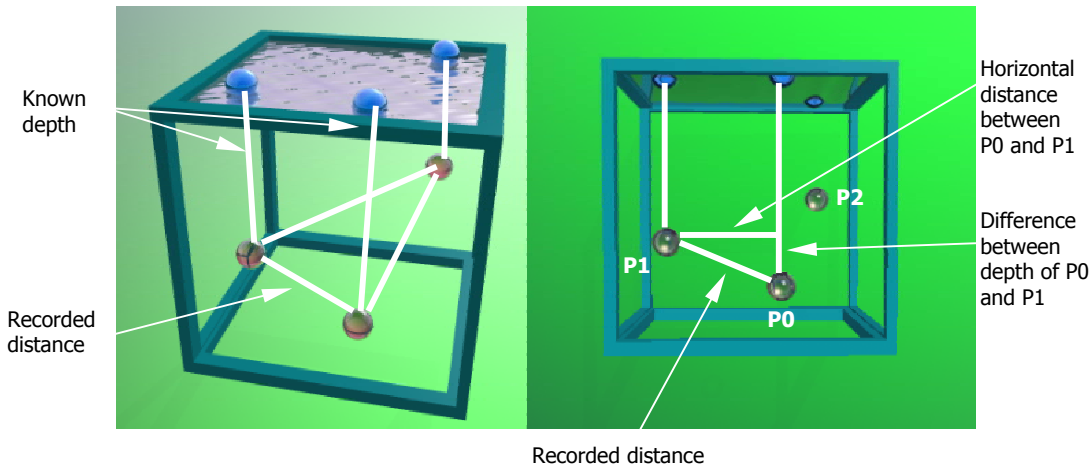


Figure 12. Relative positions of 3 robots in 3D.

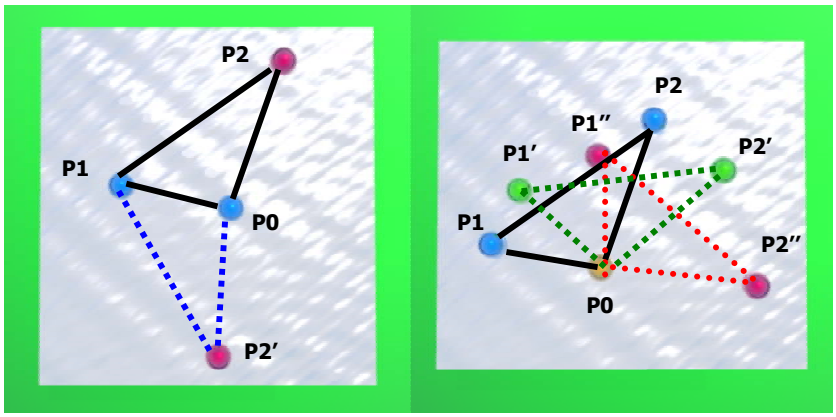


Figure 13. Viewed from top. Left: P2 flipped around P0-P1 line. Right: Robots rotated around P0.

according to data. The *relative estimation algorithm* then calculates the triangle for the third robot using these two robots. The position of other robots can then be calculated one triangle at a time. Once the position of a robot is calculated, it can be used later on in the calculation of other triangles. The heuristic is used to judge between the two possible solutions caused by flipping.

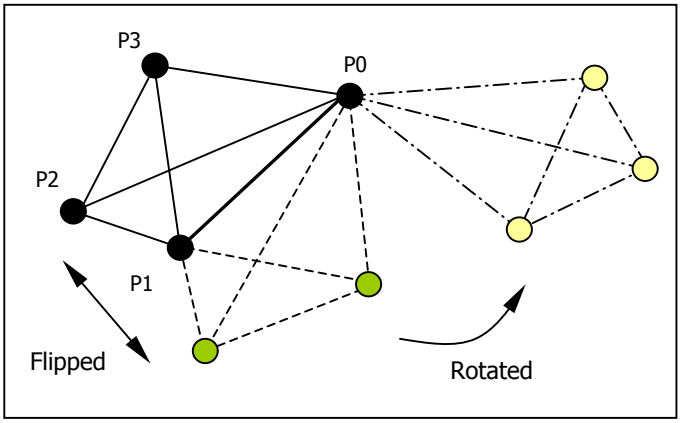


Figure 14. There is still an infinite number of solutions with four robots. Top view.

To solve the position of a robot in the general case, consider the triangle shown in Figure 15 as the possible triangle used for calculation of the positions. This is when the robots are viewed from top and basically their position mapped into $y = 0$ plane (ocean surface). The idea is to calculate the position of P2. The three sides a, b and c are known. The position of P0 and P1 is also known. Using the cosine law it is possible to calculate the angle between P0-P1 line and P0-P2 line. Using simple geometry it is then easy to calculate two possible solutions for the position of P2. The position that is closest to the previously calculated point is selected.

Special cases

In the course of the calculations there are a number of special cases that can occur which prevents the algorithm to find a precise answer for a particular time step. To have a better accuracy, the algorithm tries to detect these special cases as much as possible, so the right choice for the location of the robots can be found. In worst case scenario, if the algorithm has no way to calculate the location of a particular robot, the location of the robot at the previous time step is used instead. Some of the special cases are as follows:

A. The first special case is when the three robots are in the same line. The triangle solution would not hold anymore and the algorithm should detect this as a special case which is the same as solving in 2D. As shown in Figure 16, the precise (but relative) position of the third robot can be calculated without using the heuristic. The real distance between P0 and P2 which is c can be used to select the best option out of HP1 and HP2.

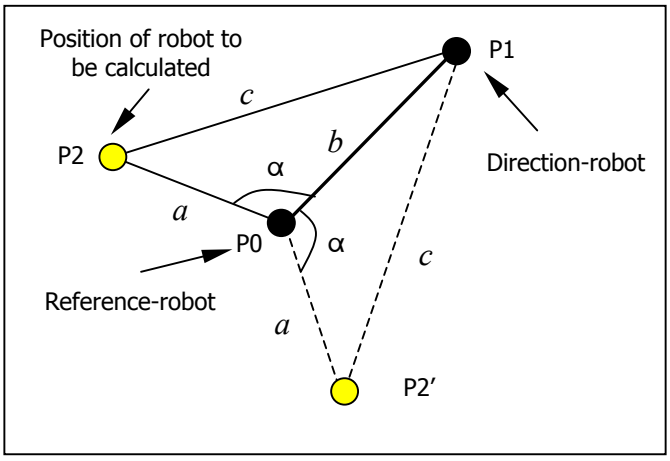


Figure 15. The flipped (dotted) triangle in Y=0 plane which is to be solved.

B. Consider the triangle illustrated in Figure 15. The idea is to calculate the position of robot P2. However there are two special cases when $a = 0$ or $c = 0$, which means that P2 for the first case can be the same as P0 in $y = 0$ plane, but can have a different depth. For the second case it can be the same location as P1 but again with its known depth.

C. However the case for $b = 0$ is a whole different scenario. In this case the initial rotation is lost. This means that P2 can be any position on the circle around P0 with a radius of the known distance between them. For this case it is possible to use the previous position of P2 in the last time step and "predict" the new position of it as if it was moving in an straight line with the same speed. This line has two crossing points with the circle. Of the two possible solutions, the answer is the point that is closest to the predicted point. This method will not always give a precise answer, but it is a relatively good solution.

D. Another special case is when P1 is crossing right through the position of P0. This means at one time step, P1 is a small distance away from P0 and in the next time step it is a small distance on the other side of P0 as P1 only moves in a particular direction. Using the distance value itself or even the heuristic it is not always possible to decide between the two possible solutions. This special case can be detected by observing the position of other robots. When P1 is moved to the other side, as the direction is still kept constant, all other probes will be "flipped" to another location. This flipping can be detected and from then onwards the algorithm will turn all other robots 180 degrees to compensate for this event. This can happen several times as P1 might cross P0 many times during a particular sampling.

E. Suppose there are n robots and n-1 are in the same line and the last one is crossing from one side of this line to the other side. This is again the problem of symmetry, since there are two possible solutions. If the nth robot is just crossing the line, the two possible solutions are very close together, and difficult to choose between. For this reason the predicted position of the nth robot is used to select between the two possible solutions.

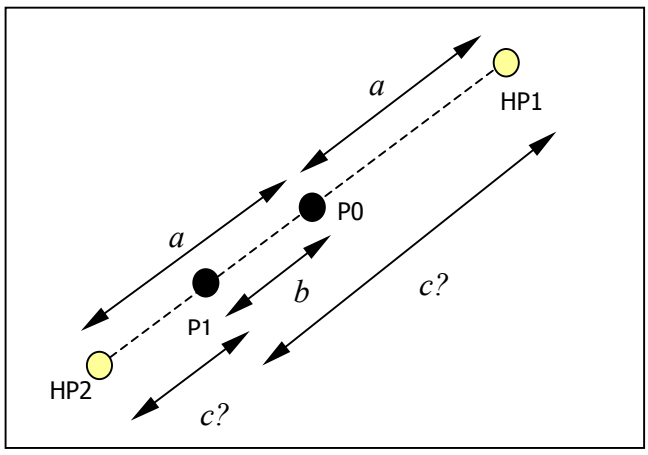


Figure 16. A special case: Top view of the three probes in a line. The solutions in 2D are indicated by (c?). See text for details.

Output of relative estimation algorithm

The result of *relative estimation algorithm* for a particular simulated example in 2D is shown in Figure 17. What is shown is the traces of the robots and the robots themselves at two

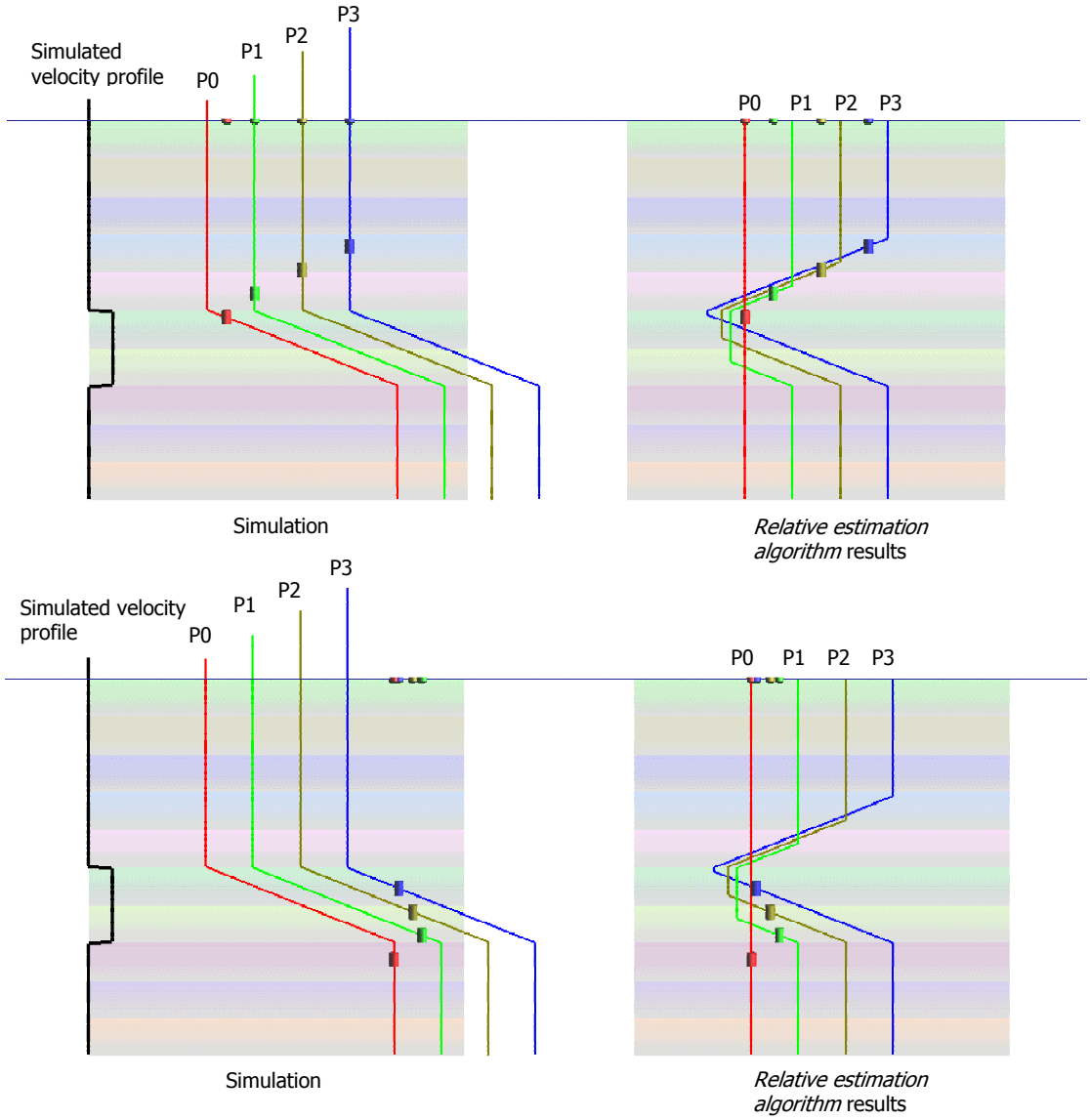


Figure 17. Demonstration of 2D velocity profile estimation algorithm. Top Row: P0 has entered the current, P1 is not in yet. Bottom row: P0 is now out of current and P1 is still in current.

different time steps. At each time step the calculated relative shape of the flock is the same as the simulated shape. The only difference is the limitation caused by the constant location of reference robot. An example in 3D is shown in Figure18 using a simulated linear current. Here the effect of the direction-robot is much more pronounced, which is why the trace has the circular shape. If the orientation of this robot is changed due to the currents, the effect is ignored and so the entire robots are rotated to keep the shape consistent with the distance information. In any case, the relative shape of the flock is always calculated correctly.

As experience showed, using four robots is better than using only three robots. When the heuristic is used, it is better to use previously calculated positions of two robots than only one to decide between the possible solutions. For more robots, the algorithm has a choice of selecting previously calculated positions of any two robots to use in the triangle to solve the position of the next robot. Since special cases can still occur, the algorithm first tries to find the best pair that make a triangle with the robot, otherwise it continues with other permutations to find the set of robots that can help locating this robot under more limited situations. If all fails, the algorithm can still use the techniques from the above special cases to solve the situation.

8. Calculating absolute positions of robots and velocity profile of the currents in 2D

The output of the *relative estimation algorithm* is of course relative. The next step is to find the absolute positions of the robots and the velocity profile of the currents that each robot experienced. The *2D/3D velocity profile estimation algorithm* and *absolute estimation algorithm* work hand in hand to solve this part, which is perhaps the most difficult part of the whole process. Interestingly, as it turns out we need to add a few constraints to the problem to solve it. A few simple but very effective requirements are all we need to reduce the problem from a very general case to a special case where we can take advantage of the requirements we just added. The case for the 2D environment can be solved in a simpler way in comparison with 3D environments. The technique for 3D however, can also be used for 2D environments.

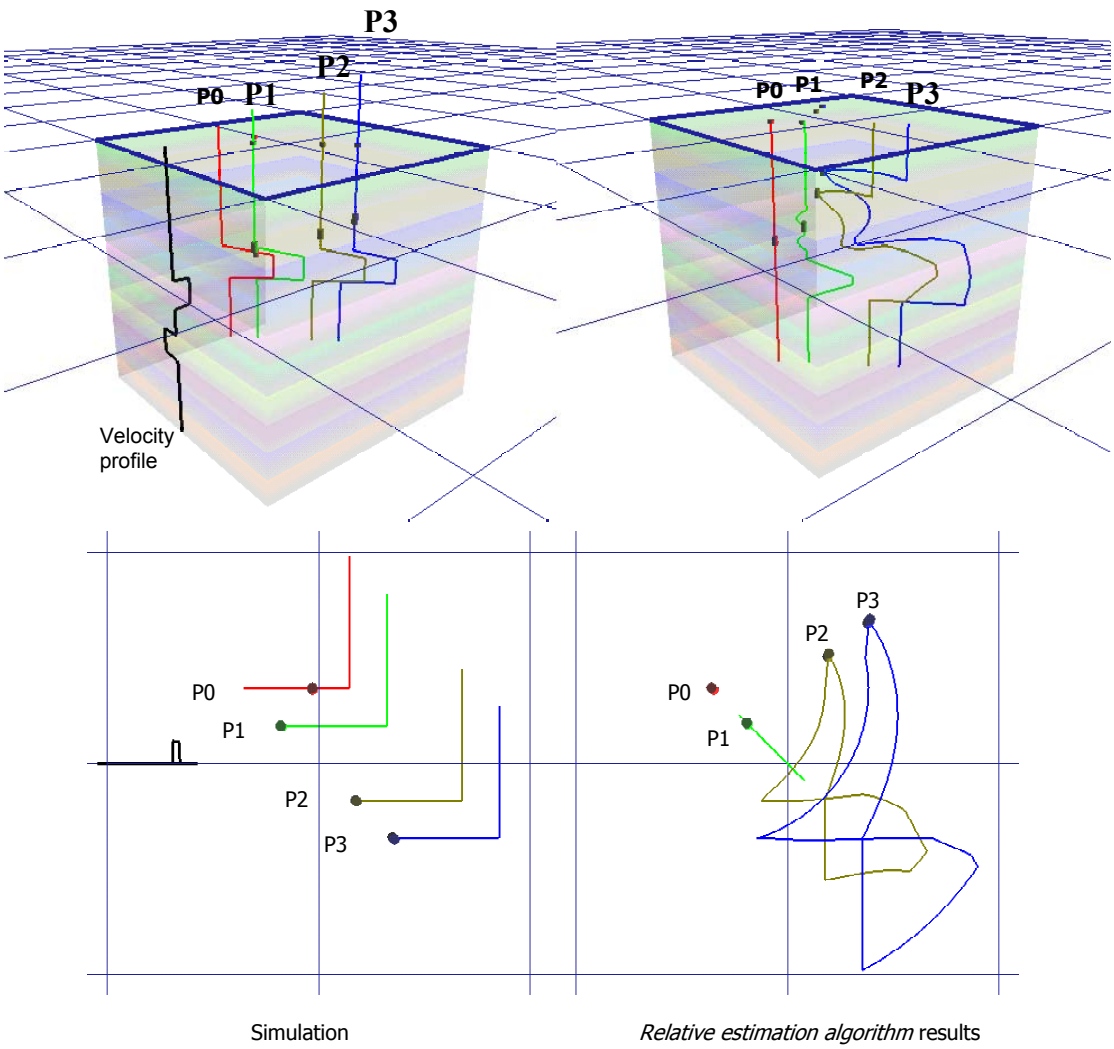


Figure 18. The left hand side figures show the simulation and the right hand side figures shows the output of *relative estimation algorithm*. The top row is at time step 0 before the robots are dropped into ocean. The bottom row is some time later when reference robot is experiencing a current, and this is viewed from top. Notice that the flock is estimated with right shape, except that the rotation of the flock is different. The position can also change as the flock goes deeper.

2D Velocity profile estimation algorithm

The key to find out the velocity profile of all robots is to first find out what is the velocity profile of the reference robot. Once this is known there are three steps to follow:

1. By using the velocity profile of the reference robot and its original position, calculate the absolute position of this robot over time.
2. Then, by using the relative position of all other robots, calculate the absolute position of all other robots. (*Absolute estimation algorithm*)
3. Now, using the absolute position of all other robots, the velocity profile which is practically the rate of change of their position over time can be calculated. The set of velocity profiles of all robots provides the ultimate solution.

To find out the velocity profile of the reference robot, a set of requirements are needed. If satisfied, then using the technique described below, this velocity profile can be calculated. The idea is to use another robots movements to calculate the velocity profile of the reference robot. As an example, the two robots can be P0 (robot 0) and P1 (robot 1) as shown in Figure 17.

Requirements for 2D

1. Original point for robot 0 and robot 1 should be different. Otherwise direction is lost.
2. Depth of robot 0 should be always lower than robot 1. This is essential for calculation of the velocity profile.
3. Robot 0 and robot 1 should experience exactly the same velocity profile, but this does not have to be simultaneous.

The third requirement means that they can be affected by a current at different times when they reach it, but they should be affected with the same amount of disturbance once they are in it. This is not a difficult requirement to achieve at least approximately. If the robots are dropped close to each other, and if the currents do not change as much in the time that the two robots are going to pass through, then the requirement is practically satisfied.

Using this, the algorithm is as follows. Since one of the robots is deeper than the other robot at all times (direct result of the requirement), P0 in this case, the change of the distance between P0 and P1 can only be the result of P0 experiencing a current at its depth. So when it is at time step n, the rate of the change of the horizontal distance between P0 and P1 is the horizontal speed of P0 in the opposite direction. This speed is then saved in the velocity profile array for this particular depth. This calculation continues to fill the velocity profile. But once P1 hits the same current, its movement is affected both by the change of P0 and itself being in a current. But we already know the velocity profile of the current at this depth since P0 was in it earlier, and considering requirement No. 3 both should have experienced the same current. Using this we can cancel out the movement imposed on P1 and be left only with the effect left on P0. This of course can be used to calculate the velocity profile for P0 at this new depth which in turn would be saved for later use.

By the end of this routine the velocity profile of P0 is calculated. The results for the estimated velocity profile for reference probe is shown in Figure 19 along with the simulation.

There is also a requirement for the choice of two robots used in this algorithm. The reference robot should certainly be one of them but the other robot can be selected from all the rest of

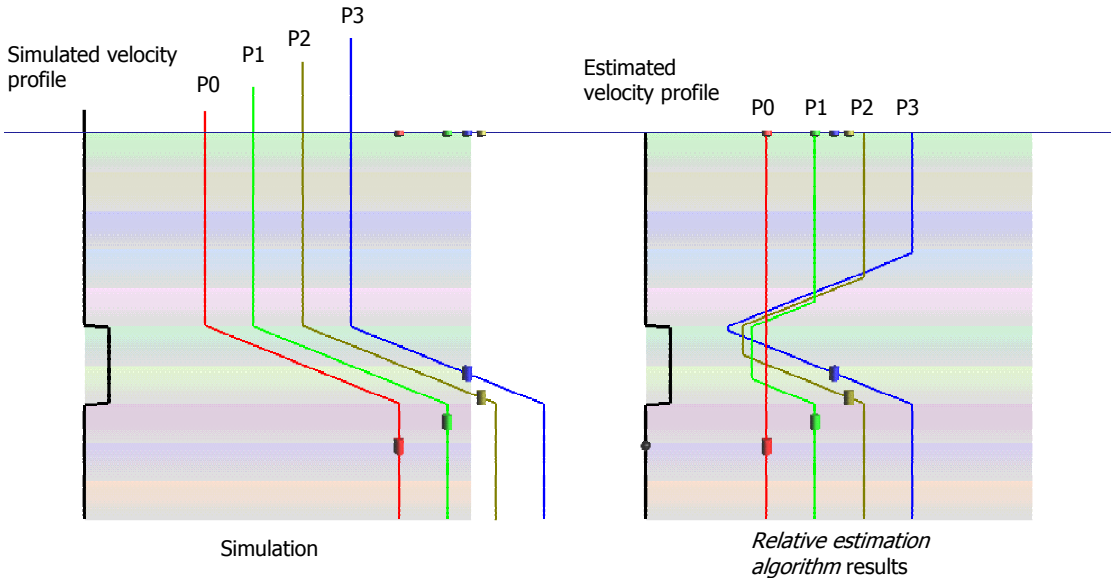


Figure 19. The comparison of the simulated velocity profile and estimated velocity profile, which is the result of *2D velocity profile estimation algorithm*.

the robots in the flock. However the choice for the reference robot is open when it is first selected in the start of *flock estimation algorithm*. So there is no limitation in this respect.

Experiments in 2D

This section contains a few examples of different currents simulated and the results obtained using the *2D velocity profile estimation algorithm* and the comparison between them.

Comparison method

For Class 1 currents, the velocity profile of all robots is exactly the same. Therefore, to compare the result with the simulation, the estimated velocity profile for reference robot is compared with the velocity profile that was setup for the simulation. If e_i is the error for measurement i and s_i is the simulation profile at i and t_i is the estimated profile at i, then the error is

$$e_i = t_i - s_i$$

This quantity is graphed for each experiment to show the accuracy of the algorithm.

There are three examples of Class 1 currents used for showing the performance results of the algorithm in the following sections. They are shown in Figure 20.

1. *Linear currents.* These are currents that are constant between two particular depths. There can be any number of them in different directions.
2. *Slope current.* This is a current that increases with a constant rate as the depth is increased.
3. *Sine wave current.* This is a current that has a velocity profile that looks like a sine wave. This is close to what real world currents look like.

The figures in this section Figures 21-23 are all in a 2D environment. The simulation is setup for each of the three kinds of currents and the velocity profile is calculated. They are then compared at each time step and e_i is then graphed (Class 3 currents, upwelling and downwelling, are not considered in this paper).

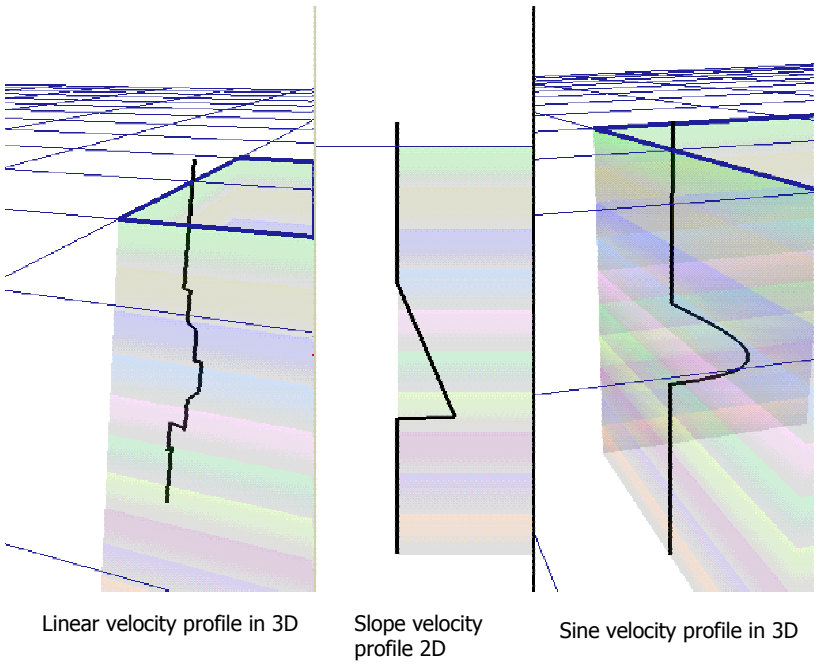


Figure 20. Three different kinds of Class 1 currents.

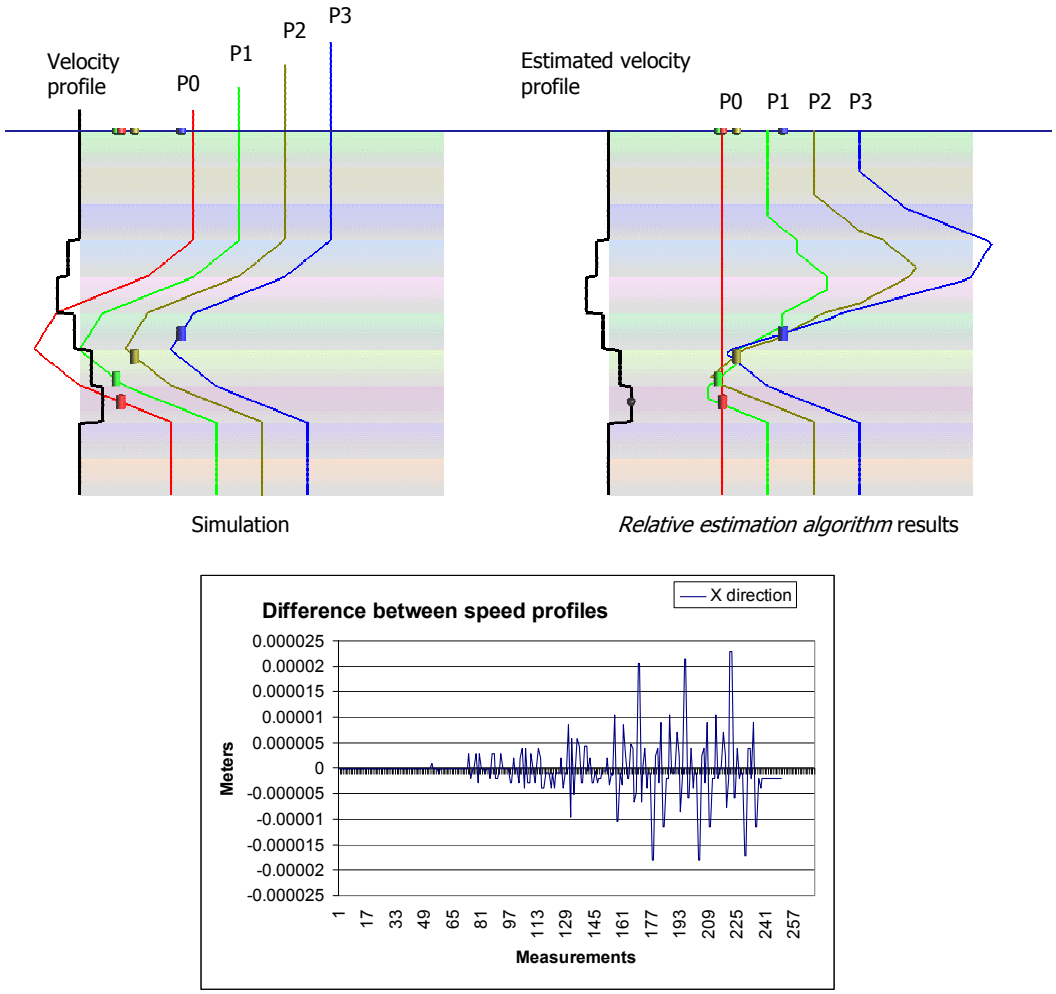


Figure 21. Linear currents. The graph shows the percentage of the difference between simulated and estimated velocity profiles.

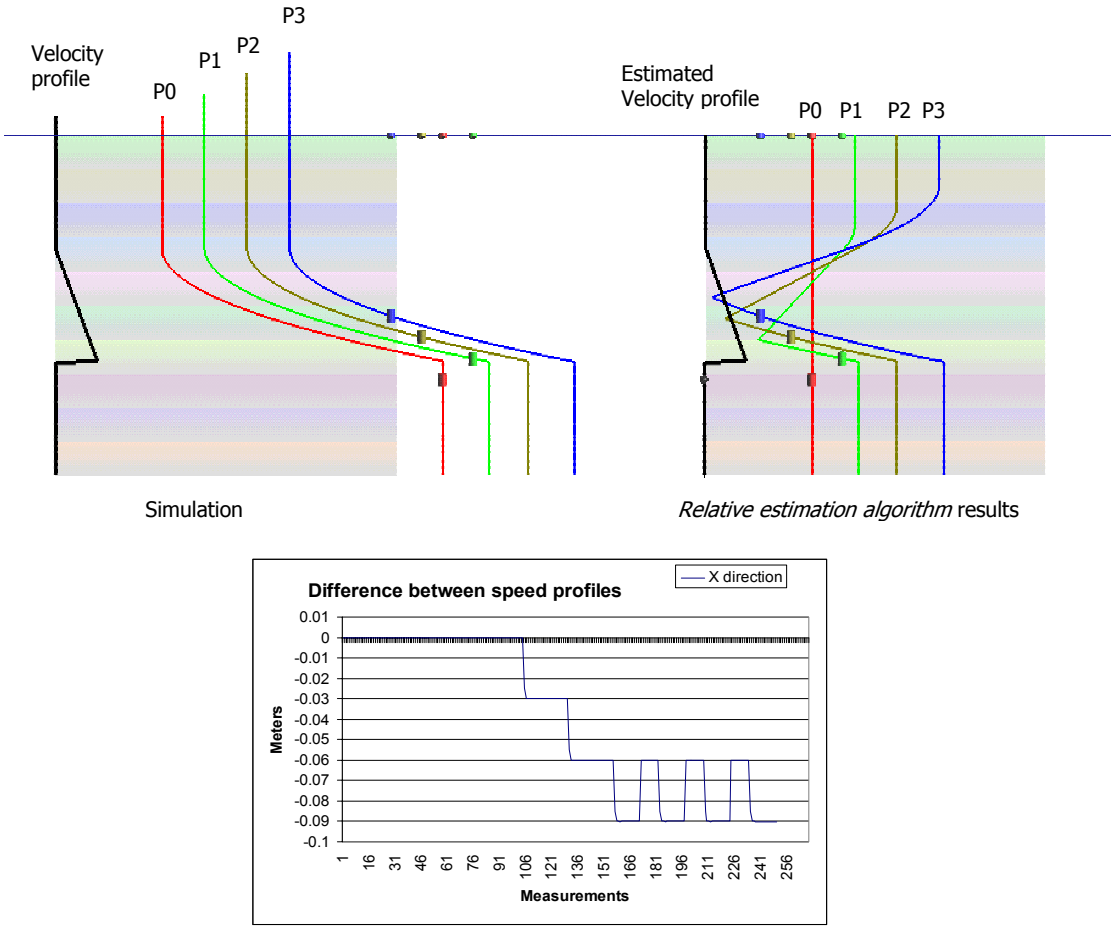


Figure 22. Slope current. The graph shows the percentage of the difference between simulated and estimated velocity profiles.

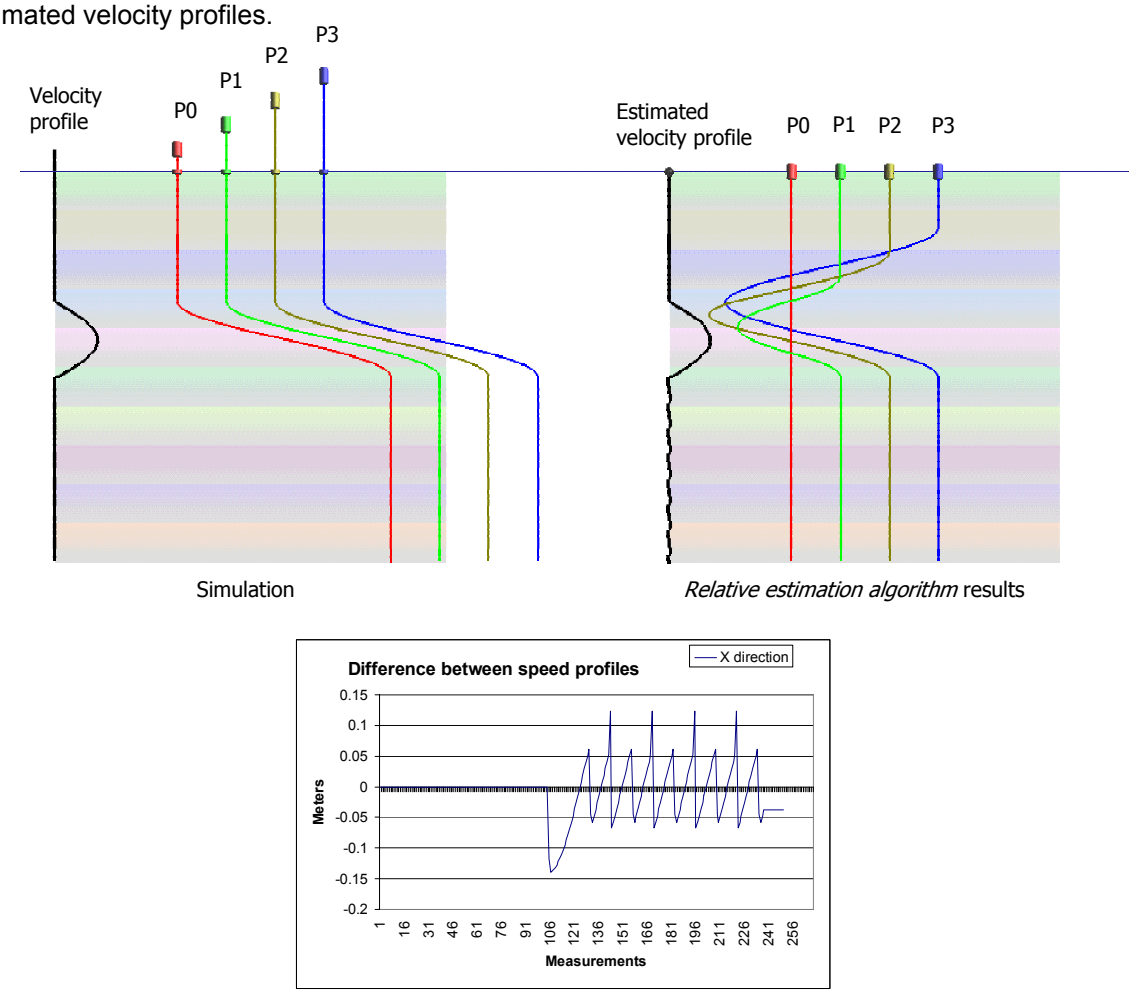


Figure 23. Sine current. The graph shows the percentage of the difference between simulated and estimated velocity profiles.

9. Calculating absolute positions of robots and velocity profile of the currents in 3D

To solve the absolute values in 3D, we can use the *3D velocity profile estimation algorithm*. The same approach of adding vital requirements can be applied in 3D but with a lot more complexity, though interestingly with the same set of requirements. Since there is an extra dimension, there is another unknown parameter in the calculated relative positions, which is the rotation of the flock. Since this orientation is kept constant, the entire flock seems to be rotating around when it hits a particular current. The algorithm is able to get rid of this effect and eventually end up with the final absolute positions of the robots.

As it turns out, in this case we need three robots to do the initial calculation. The idea is to calculate the velocity profile and the absolute position of these three robots (called *seed robots*) . Once this is calculated it follows these steps:

1. Using the three robots absolute positions and the relative position of all other robots, calculate the absolute position of all other robots.
2. Using the absolute position of the rest of robots, calculate the velocity profile for each of them which is the rate of change of their position over time. This together with the already calculated velocity profile of the *seed robots* makes the final solution.

For this algorithm to work these set of requirements should be satisfied.

Requirements for 3D

1. Original point for robot 0, robot 1 and robot 2 should be different. Otherwise orientation is lost.
2. Depth of robot 0 should be always lower than robot 1 and robot 2. This is essential for calculation of the velocity profile.
3. Robot 0, robot 1 and robot 2 should experience exactly the same velocity profile, but this can happen at different times.

3D Examples

The same set of experiments described for 2D can be done in 3D with currents in 3D using the same comparison methods described earlier detail by Honary and, and will not be discussed here for reasons of space. However it is useful to give some 3D examples at this point. The first shows two orthogonal linear currents (Figure 24). The second shows a sine current in a particular direction (Figure 25).

As it is possible to see from the graphs, both in 2D and 3D, once an error has occurred at a particular measurement it is carried over to the next measurement and is repeated until the end of the measurements. As discussed before this is the direct effect of the *2D/3D velocity profile estimation algorithm*. If the error in the earlier measurement can be removed, then the rest of the measurements would also have a better performance. The error in these experiments is due to some inaccuracy of the calculations since it is an approximation, and also due to numerous special cases happening in each experiment and that the algorithm is forced to select the “best” option available which might not necessarily lead to a result exactly the same as the simulation.

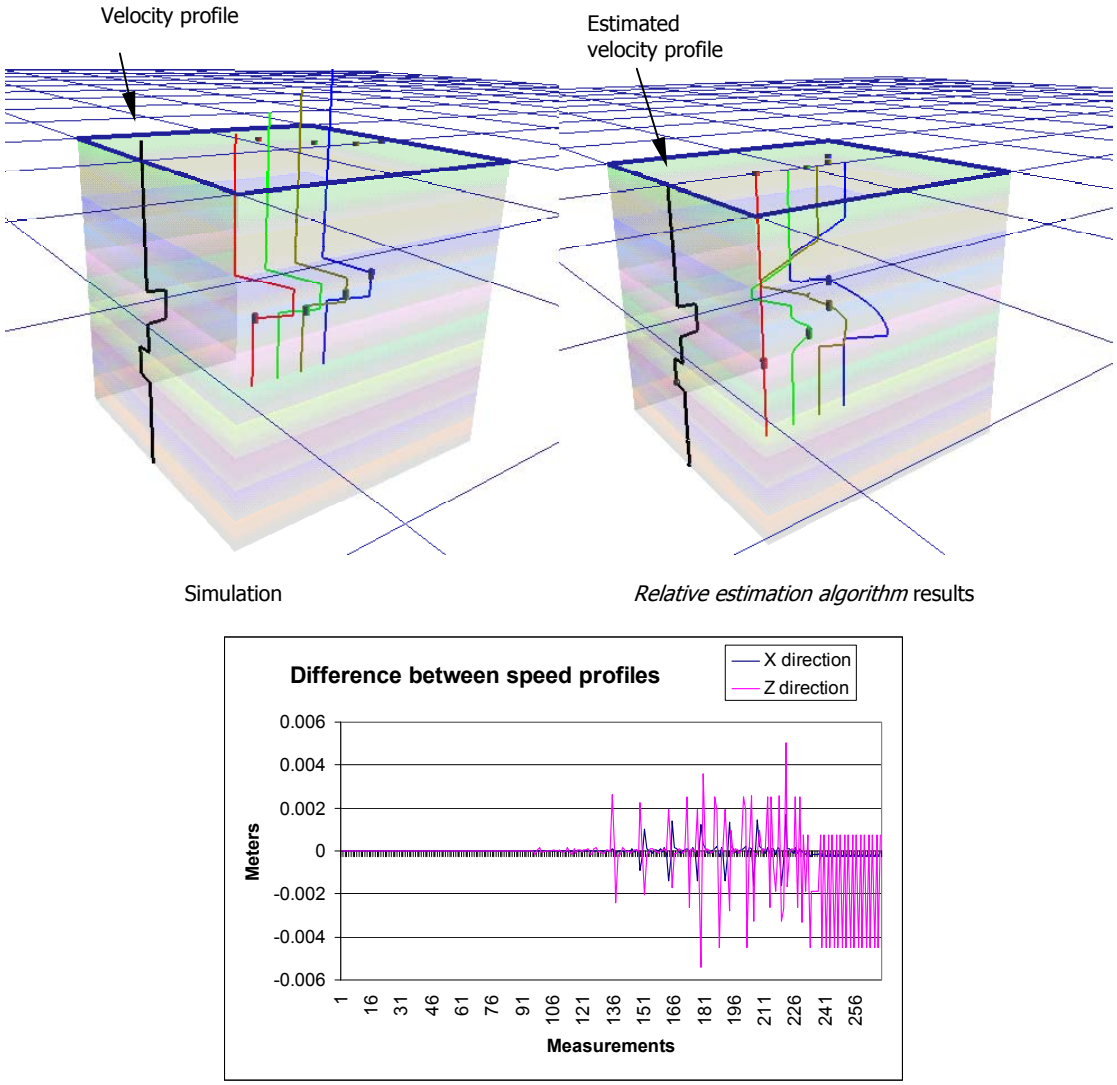


Figure 24. Linear (orthogonal) currents in 3D and the comparison between the simulated and estimated velocity profiles.

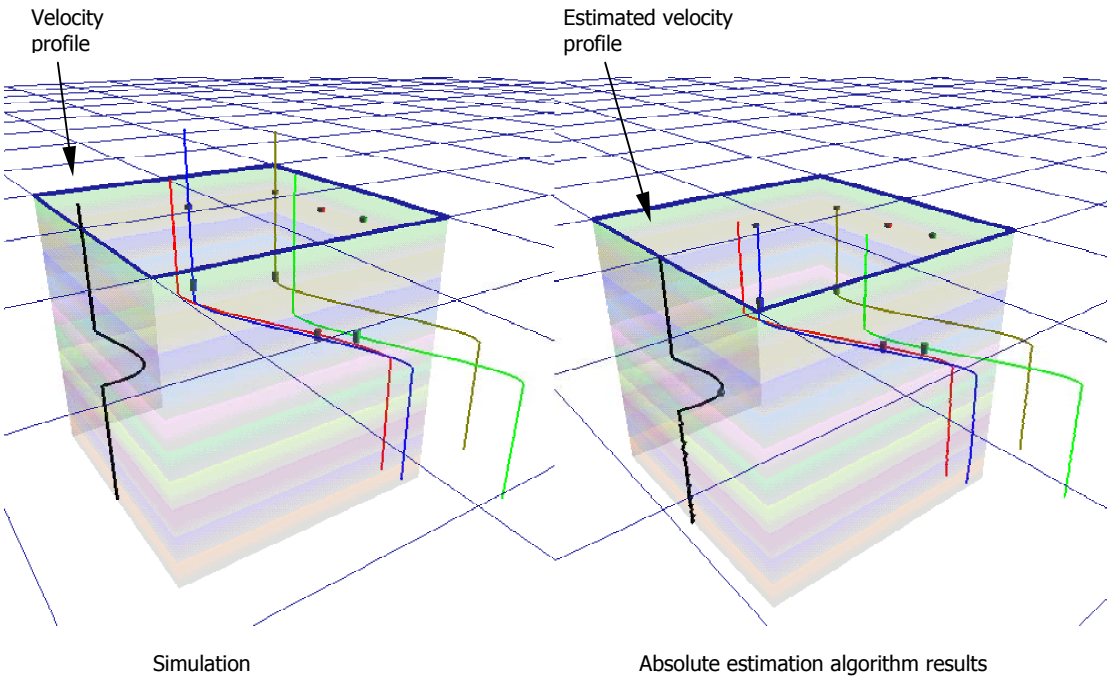
10. Discussion

In this paper we have outlined the principle of flock distortion, and shown how it might be used in mapping ocean currents. We have demonstrated the principle in two dimensions, using terrestrial robots and simulation. We have also shown that a three-dimensional simulation is possible, though not all this work is reported here.

There are, of course, many issues yet to be resolved, and our work on these problems is continuing. In particular, we are addressing the following issues:

Original formation

The first issue is what should be the original formation of the robots when they are dropped into the ocean. As imposed by the *3D velocity profile estimation algorithm* there has to be at least three robots that are close together. But how close? This depends on the overall performance of the algorithm when tested with the real ocean and real robots, which can only be estimated in future. However for time being, one solution to cover this aspect is to give the algorithm the ability to select any three robots it prefers as the *seed robots*. There can be two different possible designs for this. One is to calculate the closeness of all permutations of



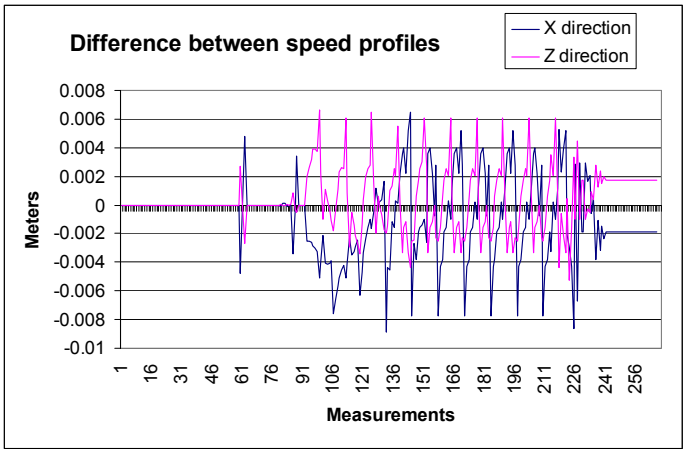


Figure 25. Sine current and the comparison between the simulated and estimated velocity profiles. The results of the absolute estimation algorithm is also shown on the right, which should be the same as the real position of the original robots.

three robots in the flock of robots and then select the three that are the closest. The second possible design is to calculate the velocity profile using a set of three robots, and then recalculate using another set until all permutations are done. Then select the best possible result, which is perhaps the most common result, as the most likely answer.

Depth variation

One of the *seed robots* should be lower than the other two. This means that at least one of them should be dropped into the ocean sooner than the others. The choice for the depth of other robots is free. Theoretically there is no difference between dropping the robots with varied depth, or dropping them all but one in a certain time and the other one sooner than all the others as an extreme example. But in practice it is better to have varied depth for the robots. If it happens that the robots experience the same exact currents, then their depth difference would stay the same. However if the deeper robot is never retrieved, then all the remaining robots might be in the same level as they descend. This means the algorithm would fail to calculate the path, since there are no longer any distance changes or a flock distortion to represent the effect of the currents.

Range

The wider the robots are dropped, the wider the area that can be sampled. However as a result it will have less resolution. This width also depends on the range that the robots can see each other, which depends on the choice of the sonar selected for them. The better the sonar, the more expensive the robots will be, and there has to be a compromise. So the maximum width of the area depends on the hardware used. As illustrated in Figure 26, if a robot (A) cannot read the distance of another robot (B) because it has gone out of range at a particular time step, then the algorithm first tries to find the distance from (B)'s point of view. However if this distance is also unknown, it is still possible to calculate the relative position of (A) using another two robots such as (C) and (D) which are closer to (A). Generally as long as the distance between a robot and two other robots with already-calculated position are known (preferably three), it is possible to calculate its position. This means that not all the robots have to see all other robots. If there is no way to know the position of a robot at a particular time step, the algorithm can simply use its position at previous time step. Since this happens rarely, it should not affect the performance of the algorithm as much.

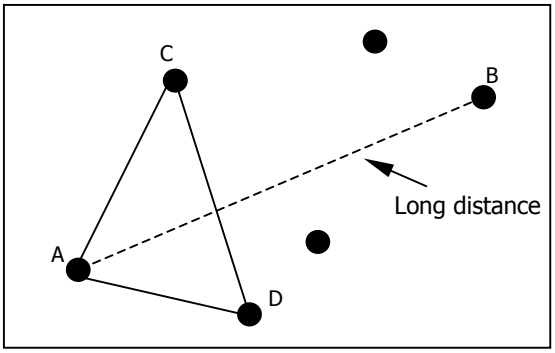


Figure 26. Robot (A) is drifting away from the flock, and cannot detect (B) anymore.

The number of robots

The number of robots needed for a particular area depends on the application. If the plan is to sample an area with high resolution, then the more robots the better. Also if a bigger area is going to be sampled, more robots are also better to keep the resolution at least the same. Of course one limitation for this is the cost of robots. As far as the algorithm is concerned, there has to be at least three robots in a flock. However, if one of them is never retrieved for whatever reason, the algorithm would fail to find the solution. The rate of loosing the robots is something highly practical and is difficult to speculate until it has been tested in reality. The performance of the algorithm is increased by having more robots since the data can be double checked for more robots and therefore the calculations can be more consistent.

To cover a particular area there are two ways to drop the robots. One is to drop all the robots as one flock in the area. The other way is to use small flocks, and drop them one at a time or all together but in different places to cover the area. Here the flocks can be set to different frequencies so they would not interfere with each other. The advantage with the latter case is reusing the robots after each sampling provided that the currents do not change over time that much. Also this way the robots can be dropped more densely in one particular area, compared to another. Thus achieving a higher resolution in the area of interest. However the second method requires a number of times of deploying the robots and this costs more than dropping the whole flock in one go. The first case however might have range problem. After all if the area is big, the robots on both ends will not see each other and as they descend,

more robots might go out of range and effectively prevent the algorithm from finding the position of all robots. Basically the first method works best for smaller unknown areas, and the second method for larger areas or "interesting" areas that can be sampled with varied resolution.

Losing robots

If a number of robots are dropped into the ocean and the collected data has some missing robots, the program is unable to estimate the position of those robots. This can happen if a robot goes down and never comes to surface, or even it comes to surface but it is unable to pass its data across. The algorithm needs to know the depth of each robot at each time step to solve the position of that robot. If only the distance information is missing, like going out of range, the algorithm is still capable of estimating the position. However if the depth information is also lost, then the position of that robot is entirely lost and could not be used at all.

One way to avoid this problem is let the robots communicate with each other when they are underwater. For this we can use the idea behind ad-hoc networks. A robot can save its information and pass a vital subset of them (such as depth) to a number of neighbours. This information can in turn travel to more neighbours and spread around. The robots have a limitation on their memory, so not every robot should end up having the entire set of data for all robots. By adjusting the number of neighbors that the robots are going to communicate it is possible to estimate the efficient amount of memory they need to save the data. Now in the event of a robot loss, the critical information can be collected from another surviving robot that happens to "know" about the missing robot and then can be used in the algorithm.

By having the ability to compensate for the possibility of the loss of robots, this method proves to be robust and less prone to real world problems.

Different classes of currents

With the algorithm described so far, it is possible to solve currents of type Class 1 and Class 2. For Class 2 currents, as long as the requirements are satisfied, it is possible to get different velocity profiles for different locations in the ocean as they are calculated separately for each robot.

Class 3 currents have a vertical element added. The vertical element of the oceanic currents depends solely on the descent rate of the robots which is precisely known as it has been recorded using the depth meter onboard the robot. The descent rate depends on the buoyancy of the robot. This is controllable by the temperature of the oil inside and hence controlling the density of the robot. If the robot's descent rate can be predicted accurately when it does not experience any currents, it is then possible to calculate the vertical element of the current as it is the subtraction of the real descent rate and the predicted rate.

As explained before, the robot can use its ability in controlling its buoyancy to try to stay with the flock if it is sinking too fast or too slow in comparison with the others. It can also use this as a method to change the resolution of sampling, or for making a particular formation. The latter can be used to make a particular sensor array. The depth variation is under the control of the robot and so it can be logged for later use. Modeling the effect of the buoyancy control of the robot in its sinking rate can turn out to be a tough challenge and is already under investigation. The more accurately the sinking rate is predicted the more accurately the vertical element of the current can be estimated.

Class 4 currents have time variation added on top. Of course if this does not interfere with the requirements of the algorithm, the solution to Class 3 currents can be used for this class of currents as well. However it is very likely that real world currents start interfering with the requirements, and the possible solution to that can only come when real data is collected and the effect of that over the algorithm has been investigated.

References

[1] Ross, D.A., (1995) *Introduction to oceanography,* Harper Collins College Publishers.

[2] Davis, R.E., Webb, D.C., Regier, L.A. and Dufour, J., *The Autonomous Lagrangian Circulation Explorer (ALACE),* J. Atmospheric & Oceanic Technology, 9(3), 264-285. (1991).

[3] Davis, R.E., *Observing the general circulation with floats.* Deep-Sea Res., 38, Suppl. 1, S531-S571. (1991).

[4] Howe, B.M., Kirkham, H., Chave, A., Maffei, A and Gaudet, S. (2001) *Wiring the Juan Fuca Plate for science: The NEPTUNE system*, Oceanology International 2001 Conference, Miami, Florida, April 3–5, (2001)

[5] McFarland, D. ,Gilhespy, I., Honary, E., DIVEBOT: A diving robot with a whale-like buoyancy mechanism. *Robotica* (in press)(2002).

[6] Melhuish, C., (1998) Collective sensing and segregation in robots with minimal sensing. *From animals to animats 5* (5th Int. Conference on the simulation of adaptive behaviour) (Eds) R. Pfeifer, B. Blumberg, J. Meyer and S. Wilson, MIT Press, 465-470. (1998).

[7] Roemmich, D., et al. (1999) On the design and implementation of Argo: A global array of profiling floats. (White papers from the Argo science team taken from http://www.argo.ucsd.edu/)