During the course of this UROP, I allowed the flock of birds to communicate with programs and services using PVM, and wrote a rudimentary program to capture data from the flock of birds and the mental model positioning system to allow for comparisons. Doing this involved multiple steps.

First, getting the flock to work. Currently there are three birds. They are configured according to the Flock of Birds Manual\(^1\). With one (small) transmitter attached, and three sensors. The largest problem with getting the flock to work was realizing that a single pin had to be cut off of the serial cable; something that did not seem to be documented. If the pin is not cut, the flock will repeatedly send a DELAY command through this pin. Also, the most common problem with the flock when the flock is set up is interference. Sometimes, this interference will result in an error message being displayed by the bird to which the transmitter is attached (a bird displays an error by flashing its LED the number of times corresponding to the error number, with error numbers listed in the flock manual). At other times, this interference will not generate an error, but will instead throw off the data. There are four main sources for this interference. First, there seems to be a large amount of metal under the floor in the media lab, so placement of the transmitter on the floor would often result in an error. Second, the metallic table created interference\(^2\). Third, Ripley itself can create interference with the flock. Lastly, any active wire placed very close to either the transmitter or sensor seemed to significantly throw off the data. The large amount of computers around the flock did not seem to have a significant effect on the accuracy when testing.

Serial communication with the Flock of Birds can be quite complicated. Rather than creating new libraries for communication, I used libfob \(^3\) (current version 1.0.2). Libfob is a C++ interface for communicating with the flock that is able to get the data from a serial port, translate the orientation from a matrix to an array of three

\(^1\) A copy can be found at:

\(^2\) A graph was made of the interference patterns of the original metallic table, and can be found at
http://www.mit.edu/~soney/brror.jpg where Ripley is facing placed at the bottom.

\(^3\) The project homepage for libfob is at http://fob.sourceforge.net/
elements representing the angles of orientation, and converts most data from the 
flock into C++ data structures.

The code for the flock of birds packets can be found in
~/src/ripley/pvm/pvmhandler where ~ is Ripley's home directory. First, there is 
singlepacket.h, which contains the code for each individual data unit from the flock of 
birds. Each of these consists of an id, as an integer; three position elements for x, y, 
and z, which are all floats; three angle measurements, which are all floats; a 
character for a button (which is not currently used, but allows for future expansion); 
and a timewrap element. Each of these has the ability to pack and unpack itself. 
Flight packets, however, are not these singular packets; they are arrays of them. The 
code for multiple flight packets can be found in flightpacket.h. They contain a vector 
of singleflightpacket elements and an integer for the number of flight packets there 
are. When asked to pack or unpack, it asks each singleflightpacket to pack or 
unpack.

As far as constants go, PVMCommonHeader.h (located in ~/src/ripley/pvm) 
was modified to include two constants; one for the FLIGHT_PACKET group and 
another for the actual packet. In it, FLIGHT_PACKET_MESSAGE_TYPE was defined as 
2200. Also, PVM_FLIGHT_GROUP was defined as "PvmFlightGroup".

Most of the code for the flock of birds is located in 
~/src/ripley/ripley_core/fob. In this directory, args.h, commandoptions.cpp, and 
commandoptions.h are all for use by flight. Flight is the program that takes in data 
from the flock (using libfob) and then outputs the data using PVM. Much of it is 
modeled on example programs provided with libfob. In it, it surveys each bird and 
asks it for its data. It then creates a single flight packet whose vector of 
singleflightpackets is as large as the number of birds (so that the data from every 
bird at any given time should be in a single flight packet). This is mainly for 
efficiency and traffic reduction purposes. These packets are sent to the 
PVM_FLIGHT_GROUP group and are of type FLIGHT_PACKET_MESSAGE_TYPE.

Test_receiver is a test receiver I made for the flock data, which simply 
outputs the data of everything it gets. Although it is not used, it serves as a good 
example program.

Fob_mm_comparator contains the code to receive both flock of birds and 
mental model packets, but is no longer used. This is because the buffer for PVM was 
eaten up when trying to receive flight packets, so it ended up getting very few 
mental model packets. For this reason, the code was split up into two files, both of 
which are derived from fob_mm_comparator.
First, there is fob_comparator, which gets the data from the flock of birds and outputs it to FOB_DATA (each time fob_comparator is run, it deletes the data out of FOB_DATA. The data in FOB_DATA is arranged as follows: each new line represents a different bird’s data at a different time. It contains nine elements separated by spaces. From left to right, the data is the time the packet was sent in milliseconds, the current time in seconds, the bird’s id, the three position elements, and the three orientation angles.

The second file is mm_comparator, which takes the data sent via PVM from the mental model and outputs it to MM_DATA. The data in MM_DATA contains 23 elements, each separated by spaces. Each line represents a different object in the mental model at a different time. Each line contains the following elements from left to right: the time that the mental model packet was received, the time in seconds that the packet was received, the object’s ID, the object’s internal ID, the three position elements, the rotation matrix (in 12 elements), the color (in three numbers representing red, green, and blue), and the radius of the object.

One way in which the data can be used is for comparisons between the data for positions of objects between the mental model and the flock of birds. This can be done by putting the sensors for the flock into objects that can be detected by the mental model, and then comparing the data between them. For example, I placed the sensors in colored cups using duct tape, as shown in the figure to the right. The tricky part, however, is finding the correspondence between the axes for the flock of birds and the mental model. This requires some data analysis. The data analysis can be done using MatLab.

In the future, the main change that needs to be made is to allow for automatic data comparison between the flock and the mental model, and automatic correspondences between the axes. Also, allowing for automatic correction for error given by the metal table and Ripley would be helpful, but does not seem practical for two reasons. First, the type of error given (how far off in each dimension the metal throws it) may change given the position of Ripley. Secondly, the fact that the error is specific to each table might make the error correction too specific. Also, another

---

4 Not the time that the packet was sent, as is the case with the flock data. This is subject to change, however, when the mental model packets contain a TimeWrap element that says when they are sent.
5 One possible .m file is at http://www.mit.edu/~soney/fob.m
possible improvement would be to use wireless sensors, rather than wired ones. This might make the system more flexible.

\[\text{One way to get around this might be to have some sort of error data file, where the error data for a particular table is inputted, and the comparator reads from whatever file represents the current table being used}\]