Running the System

Open the Unity project by selecting the UAVBehavior folder.

There are three sets of settings you can manipulate.

-Individual UAV settings

-Individual Crowd settings

-UAV spawn and Crowd spawn settings

Once inside Unity, to manipulate individual UAV settings, click Project tab -> Assets folder -> SpawnedUAV. In the Inspector tab on the right, you should see a set of parameters under the "Block Crowd 2" heading. This is the primary script for the UAV and contains our system architecture. It has several parameters, as explained below. Distances are in Unity simulation units; consider 7.5 units equal to 12 feet. All potential field strengths must be in the range [0,1]. The settings are a "prefab" so they can be spawned in multiples (credit goes to Wang/Zhou for discovering and using this functionality in their project).

To manipulate the individual crowd settings, click Project tab -> Assets folder -> SpawnedCrowd. You should see a set of parameters in the inspector tab on the right. The settings are a "prefab" so they can be spawned in multiples (credit goes to Wang/Zhou for discovering and using this functionality in their project).

To manipulate the multiple UAV spawning and multiple crowd spawning settings, click on the front wall that says "Hello World." You should see a set of parameters in the inspector tab on the right. Under the Spawn Crowds script you can set the number of crowds generated per interval, and you can set the interval. Under the Spawned UAVs script you can set the number of UAVs.

Crowds spawn in random locations and are "scared" (turn around) of motion towards them depending on the speed of motion. Crowds despawn after passing the block line (in case of failure) or after turning around.

The parameters for BlockCrowd2 are as follows:

Sim Detect Crowd Range	Maximum distance at which the simulated crowd detection sensor can perceive a crowd.
Approach Range	If any crowd is closer to the robot's plane of operation than this, the **Approaching** behavior is activated.
Threaten Range	If any crowd is closer to the robot plane of operation than this, the **Threatening** behavior is activated.
Wall Avoid Strength	The maximum strength of the uniform potential

	field perpendicular to the wall.
Wall Avoid Depth	The wall avoid field is exponential; at this distance, the field reaches 1% of Wall Avoid Strength; it does not appear beyond that point.
Crowd Avoid Strength	The maximum strength of the repulsive potential field around a crowd.
Crowd Avoid Depth	The crowd avoid field is exponential; at this distance, the field reaches 1% of Crowd Avoid Strength.
Neighbor UAV Avoid Strength	The maximum strength of the repulsive potential field around a UAV.
Neighbor UAV Avoid Depth	The neighbor UAV avoid field is exponential; at this distance, the field reaches 1% of Neighbor UAV Avoid Strength.
Keep Height Strength	The maximum strength of the potential field which maintains a certain UAV altitude; this is experienced at the surface of the floor and ceiling.
Hold Position Strength	The maximum strength of the potential field which directs the UAV to the center of the hallway during **Watching**; experienced at the left and right walls.
Follow Strength	The maximum strength of the potential field which directs the UAV toward the crowd it is attempting to follow. Experienced at full strength when left or right of the crowd by 70% of hallway width.
Rand Threaten 2DStrength	The maximum strength of the potential field which moves the UAV erratically to scare a crowd away during **Threatening**. Experienced when the UAV is directly in front of the intended crowd at chest height.
Rand Threaten 2DDepth	The magnitude of the threatening potential field exponentially decreases away from the crowd's projected point onto the UAV's plane of operation. At this distance from that point, the field is at 1% Rand Threaten 2DStrength.
Rand Threaten 2DInterval	This is the duration of a particular random movement, after which a new direction is chosen.
Sim Model Force	This is the force the UAV will generate to act out a potential field of strength 1.0; it scales the potential field result of all active motor schemas.

Development Environment

To access a script, go to the Project tab (lower left by default), and select All Scripts under favorites. Double click the script you wish to debug/edit. This opens MonoDevelop-Unity, and loads the script project with the script you clicked opened for debug/edit. You can edit here and have changes reflected in Unity, or click the play button in the top left in order to attach to Unity and perform integrated debugging while the simulation executes.

The BlockCrowd2 script provides a “vector sum” object displayed for debugging. You can find this in the scene tab while running the simulation. As noted in the Console tab’s log, the blue lines are **KeepHeight** potential field responses, the green correspond to **Follow** or **HoldCenter**, the yellow are from **ThreateningRand2D**, the red are from **Avoid**, and the white is the summation of all the response vectors and represents the instantaneous velocity of the UAV.

ManualPersonMove.cs

This is the automation script for “Black”, the crowd which can be directly controlled by the user via the keyboard arrow keys. It contains a very simple MonoBehavior (Unity script) which turns key inputs into Translate calls to make the simulation’s crowd cylinder move.

Classes of BlockCrowd2.cs

Pass-By-Reference Wrappers:

.NET doesn't allow pointers, but we wanted to be able to provide percepts to motor schema and behaviors in such a way that we didn't have to constantly copy from the percept to each destination's percept input. These classes allow us to wrap pass-by-value percept types in pass-by-reference classes, enabling this pointer-esque behavior.

Vector3RefWrap

Wraps one of Unity's Vector3 data types, which holds coordinates of crowds, the Hokuyo sensor, etc.

FloatRefWrap

Wraps a simple float, which is used for wall percepts

Sensors:

CrowdDetectionSensor

This is a stub for a computer vision algorithm which would perform crowd recognition from color video input. This stub instead uses **GameObject.Find()** to get the crowd Game Objects from Unity and populate a list of crowds (**simCrowds**). From there it calculates a corresponding egocentric 3D vector (pointing from the UAV to the crowd) for each crowd, which it adds to **sensedCrowds**.

The public method CrowdsDetected() returns sensedCrowds.

Hokuyo

This is a stub for an algorithm that would transform actual Hokuyo sensor data (point clouds) to a single point location for each UAV. The stub instead uses each UAV's point location in the Unity environment. With this point and the known point location of the Hokuyo sensor (the sensor is 6' high on the right wall), this class calculates an vector pointing from the Hokuyo sensor to each UAV. Note that these vectors will change depending on where the UAVs are. A list of these vectors is returned by the **robotsDetected**() method.

LocalizationSensor

This is a generic stub for individual UAV egocentric localization. The location of the UAV is maintained by its position relative to the Hokuyo sensor. **UAVLocation**() returns a vector from the Hokuyo to the UAV.

Percepts:

All percepts provide information in egocentric coordinates to the rest of the system. The only part of our code that is not egocentrically oriented is the code in the sensor classes. Additionally, the perspective used is looking from behind the robot down the hallway toward the crowds. Keep in mind that from that perspective, the x-coordinates increase to the left and the z-coordinates increase backwards (in the 6 o-clock direction).

CrowdPercept

This class uses the **CrowdDetectionSensor** to generate a usable percept. It gets the list of crowd vectors (**sensedCrowds**) from **CrowdDetectionSensor** and sorts them according to crowd proximity such that index 0 refers to the closest crowd (it excludes from the list any crowds behind the UAV). The new sorted list (**BlockableCrowds**), along with the unsorted **sensedCrowds,** are the percepts to be used by behavior and motor schema classes. These percepts are updated at every frame via the **Update()** method.

LocalizationPercept

This class uses the **LocalizationSensor** class to generate a usable percept. The percept provides egocentric distances between the UAV and the two walls, ceiling and floor. This class uses the location vector (from **LocalizationSensor.UAVLocation()**) to calculate the locations of the **leftWall**, **rightWall**, **ceiling**, and **floor**. Since these values are egocentric, they change as the UAV moves around. The values are updated every frame by the Update() method.

UAVPerceptSchema

This class uses the **Hokuyo** sensor to generate a usable percept. It gets the list of UAV vectors (**robotsDetected**()) from **Hokuyo** then subtracts the UAV's "self" from the list, resulting in a list of neighboring UAVs. The vectors in this list are then transformed to be egocentric to –this– UAV. The new list (**UAVPercepts**) will be used by behavior and motor schema classes. These percepts are updated at every frame via the **Update()** method.

Motor Schemas:

abstract UAVMotorSchema

The following motor schema classes derive from this class. All the motor schema classes share the motor response fields **responseTranslate** and **responseRotate**. The UAV currently has no rotational behavior so **responseRotate** is not being used but is left in for possible future implementation. The **responseTranslate** field is a 3D vector that represents the motor response. Although the vector is 3 dimensional, the Z component will always remain 0 in our system since the UAV must stay in the plane

of the Hokuyo. For each motor schema except **Rand2D,** this vector is calculated from the UAV's location in a p-field. **responseTranslate** is updated each frame in the Update() function.

PerpendicularExponentialIncrease

This motor schema implements a uniform p-field with an exponentially increasing magnitude (gets stronger as the UAV approaches). The **responseTranslate** vector is calculated by combining the egocentric orientation of the field (**fieldOrient**), the distances from the UAV at which the field is at its minimum and maximum strengths (**minFieldCoord** and **maxFieldCoord**), and the **peakFieldStrength** which scales the response vector. Note: **minFieldCoord** is the point at which the field is 1% of the **peakFieldStrength** while **maxFieldCoord** is 100% of the **peakFieldStrength**. When the UAV is between the **minFieldCoord** and **maxFieldCoord** is when the UAV feels the field's effect. Since the UAV's location coordinates are always <0, 0, 0>, this means when **minFieldCoord** >= 0 and **maxFieldCoord** <= 0 or vice-versa. When maxFieldCoord == 0, the UAV feels the **peakFieldStrength**.

Being perpendicular and uniform, this p-field is used for avoiding the walls, ceiling and floor. Behavior classes that use this motor schema are: **KeepHeight** and **Avoid**.

AttractiveExponentialDecrease

This motor schema implements an attractive p-field with an exponentially decreasing magnitude (gets weaker as the UAV approaches). The **responseTranslate** vector is calculated using the point of the center of the field (**position**) and the distance from that point (**capDistance**) at which the field strength is capped at 100% of **peakFieldStrength**. Beyond **capDistance**, the field remains at **peakfieldstrength**. At **position**, the field strength is 0. **Peakfieldstrength** scales the response vector.

This field has the ability to hold response vector components at 0 (via **enableX, Y, Z**). This is useful for keeping the UAV from leaving the Hokuyo plane or at a height mediated by a parent behavior, namely **Watching**, **Approaching** and **Threatening**.

This motor schema is used for following crowds or attraction to a point. Behavior classes that use this motor schema are: **HoldCenter** and **Follow**.

RepulsiveExponentialIncrease

This motor schema implements a repulsive p-field with an exponentially increasing magnitude (gets stronger as the UAV approaches). The **responseTranslate** vector is calculated using the point of the center of the field (**position**) and the distance from that point (**capDistance**) at which the field strength is 1% of the **peakFieldStrength**. **peakFieldStrength** scales the response vector. This field has a dead zone inside which the field strength is zero. This dead zone lies at a radius around **position**; the length of this radius is the field **deadZoneAroundPosition**. At this radius, the field strength is 100% of **peakFieldStrength**.

Like **AttractiveExponentialDecrease**, this field has the ability to hold response vector components at 0 (via **enableX, Y, Z**).

This motor schema is used for avoiding crowds. Behavior class that uses this motor schema is: **AvoidCrowd**.

Rand2D

This motor schema implements a uniform p-field that changes to random orientations (within the x, y plane) at an interval value **interval**. Although it is uniform, this p-field exponentially increases in magnitude around a point **position** (namely the crowd being threatened). The distance from **position** at which the field is 1% of its **peakFieldStrength** is the value **capDistance**.

Behaviors:

abstract UAVBehavior

This is the base class for all the behaviors. This allows them to be contained in a collection so that they can all be updated with the latest percepts and have an opportunity to generate new output, as well as having a common property to retrieve their output. All the derived behaviors will share the **responseTranslate** field which serves as the motor output vector as it did in the motor schema classes. Additionally, every behavior has a list of **childBehaviors** and **motorSchemas**. When a behavior class uses a motor schema or a sub-behavior, it adds that MS or sub-behavior to the appropriate list. The motor responses from each item in these lists are then summed together to yield the **responseTranslate** value.

- **ResponseTranslate**: the response of the behavior
- **childBehaviors**: sub-behaviors encapsulated in this behavior
- **motorSchemas**: motor schemas encapsulated in this behavior
- **Update**() this is overridden if need be by the individual behaviors, but contains the key process of calling **Update**() on all child behaviors and motor schema, as well as the sum coordination function for their potential fields.

Child Behaviors:

KeepHeight

This behavior maintains the UAV's altitude at a certain height, **height**. It uses two **PerpendicularExponentialIncrease** motor schemas to do so, configuring them relative to the floor and ceiling with their **minFieldCoord** at **height**.

- **height**: the distance from the floor percept
- **heightRelative**: the **height** parameter gives distance from the floor desired, but the motor schemas need that in egocentric terms. **heightRelative** is the distance from the UAV to the target **height**.

HoldCenter

This behavior directs the robot to the center of a hallway (used by **Watching**). It uses a single **AttractiveExponentialDecrease** MS with only the x-dimension component enabled and at a center point **midpoint** set to **aboveEyeLevel** (a Unity Parameter).

- **midPoint**: this is where the robot should be stationed as it watches for crowds; it is determined as a point relative to the current perception of walls and floor.

Follow

This behavior directs the robot toward a crowd object but only in the x-dimension. It uses a single **AttractiveExponentialDecrease** MS with only the x-dimension component enabled and centered on **crowdProjectOnPlane.**

- **crowdProjectOnPlane**: this translates the closest crowd's position to a point on the robot's plane of operation which is at the same height as the UAV; thus the motor schema will be attracted to this point

AvoidCrowd

This behavior directs the UAV away from a crowd. It uses a single **RepulsiveExponentialIncrease** MS with only the x-dimension component enabled and centered around a crowd object. This behavior is contained in the tactical abstract behavior **Avoid**, which is always on. In **Avoid**, one instance of this behavior is active for each detected crowd.

- **nthCrowd**: which crowd in the **BlockableCrowds** percept list to avoid
- **crowdUAVLevel**: this is the crowd position to be avoided, shifted to the current UAV height.

AvoidUAV

This behavior directs the UAV away from one neighboring UAV. It uses a single R**epulsiveExponentialIncrease** MS with only the x-dimension and y-dimension components enabled, centered around a neighbor UAV object. This behavior is contained in the tactical abstract behavior **Avoid**, which is always on. In **Avoid**, one instance of this behavior is active for each neighboring UAV.

- **nthNeighbor**: which neighboring UAV in the **UAVPercepts** list to avoid
- **neighborUAVLocation**: this is the UAV position to be avoided, with just x and y; z=0.

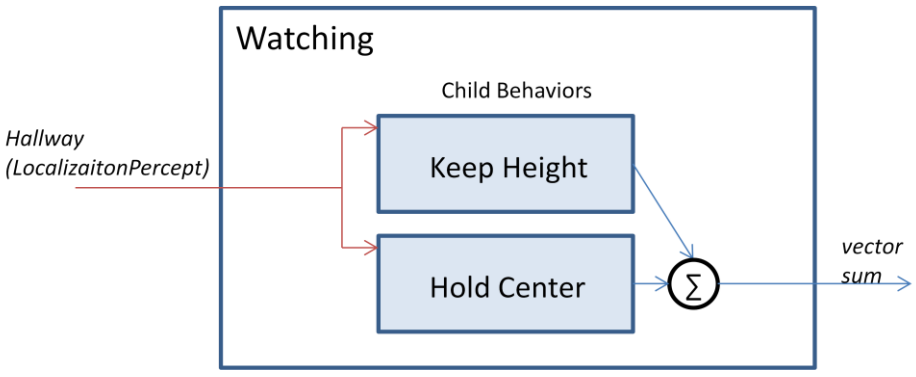
ThreateningRand2D

This behavior causes the UAV to make sudden random movements in an attempt to "scare off" an approaching crowd. It uses a **Rand2D** MS to provide the p-field that causes it to fly erratically. It passes to **Rand2D** the projection, **crowdProjectOnPlane,** of the approaching crowd's location onto the UAV's plane of operation.

- **crowdProjectOnPlane**: composed of the closest crowd's x-coordinate, the **ThreatenHeight**'s y-coordinate, and the plane of operation's z-coordinate(0). It is the position at which the **Rand2D** will be at peak strength (its **position** parameter).

Parent Behaviors:

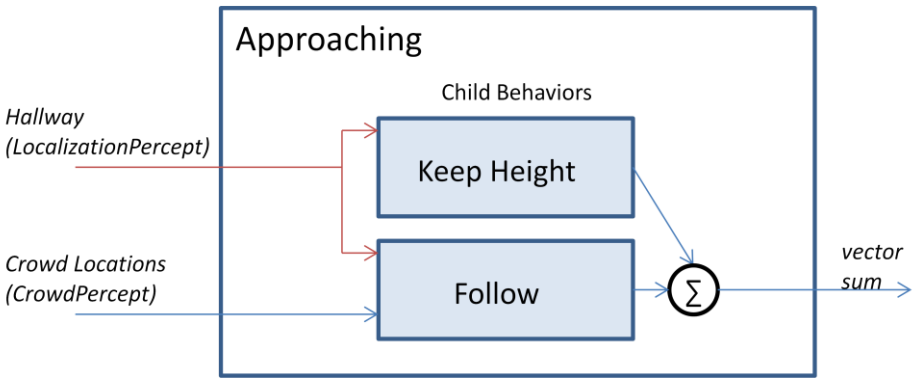
Watching



This strategic behavior directs the UAV to hold position in the middle of the hallway, above eye level. This is maintained by potential fields created in its child behaviors. This behavior is active when there are no crowds close enough to release the **Approaching** or **Threatening** behaviors. It encapsulates two child behaviors, **KeepHeight** and **HoldCenter**. **KeepHeight** is given a **height** of the value **AboveEyeLevel**, which is a public field in the BlockCrowd2 class and a Unity parameter.

- **KeepHeight**: at **AboveEyeLevel**
- **HoldCenter**

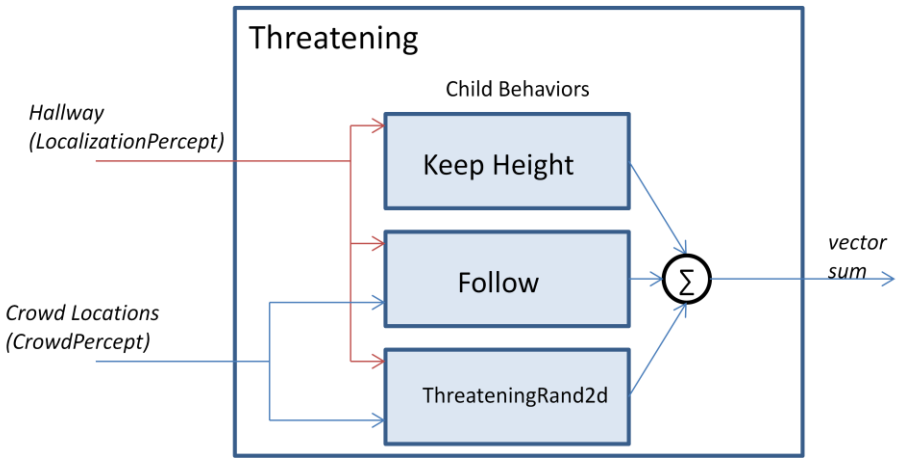
Approaching



This strategic behavior directs the UAV to descend to eye level and follow the closest crowd. This is maintained by potential fields created in its child behaviors. This behavior is released when a crowd comes within **approachRange**(a Unity parameter) but not within **threatenRange**. It encapsulates two child behaviors, **KeepHeight** and **Follow**. **KeepHeight** is given a **height** of the value **EyeLevel**(a Unity parameter). Follow is configured to use the single closest crowd as its point of attraction.

- **KeepHeight**: at **EyeLevel**
- **Follow**: attracted to the closest crowd

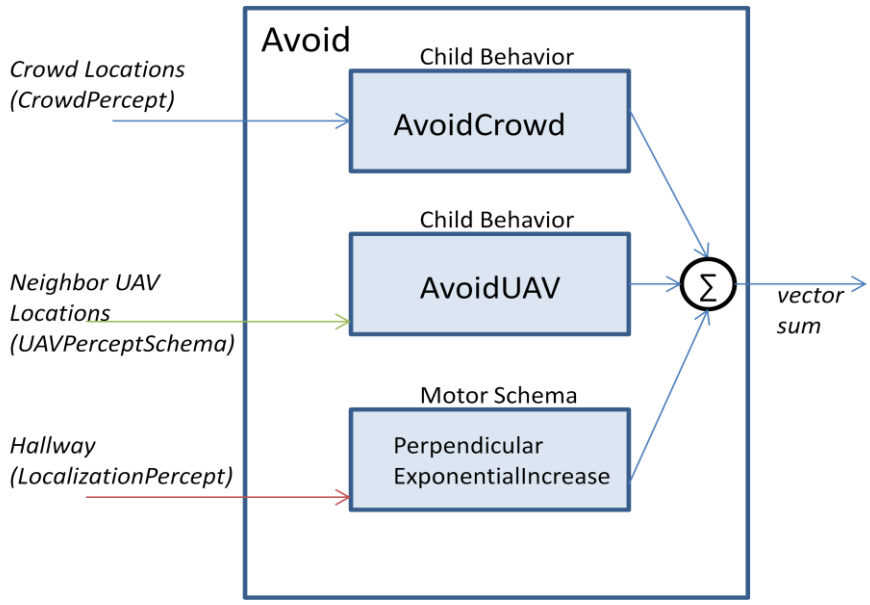
Threatening



This strategic behavior directs the UAV to descend to eye level, follow the closest crowd, and fly erratically in order to scare away an approaching crowd. This is maintained by potential fields created in its child behaviors. This behavior is released when a crowd comes within **threatenRange**. It encapsulates three child behaviors, **KeepHeight**, **Follow** and **ThreateningRand2D**.

- **KeepHeight**: given a **height** of the value **EyeLevel**(a Unity parameter)
- **Follow:** use the single closest crowd as its point of attraction
- **ThreateningRand2D**: use the single closest crowd as its point of strongest movement

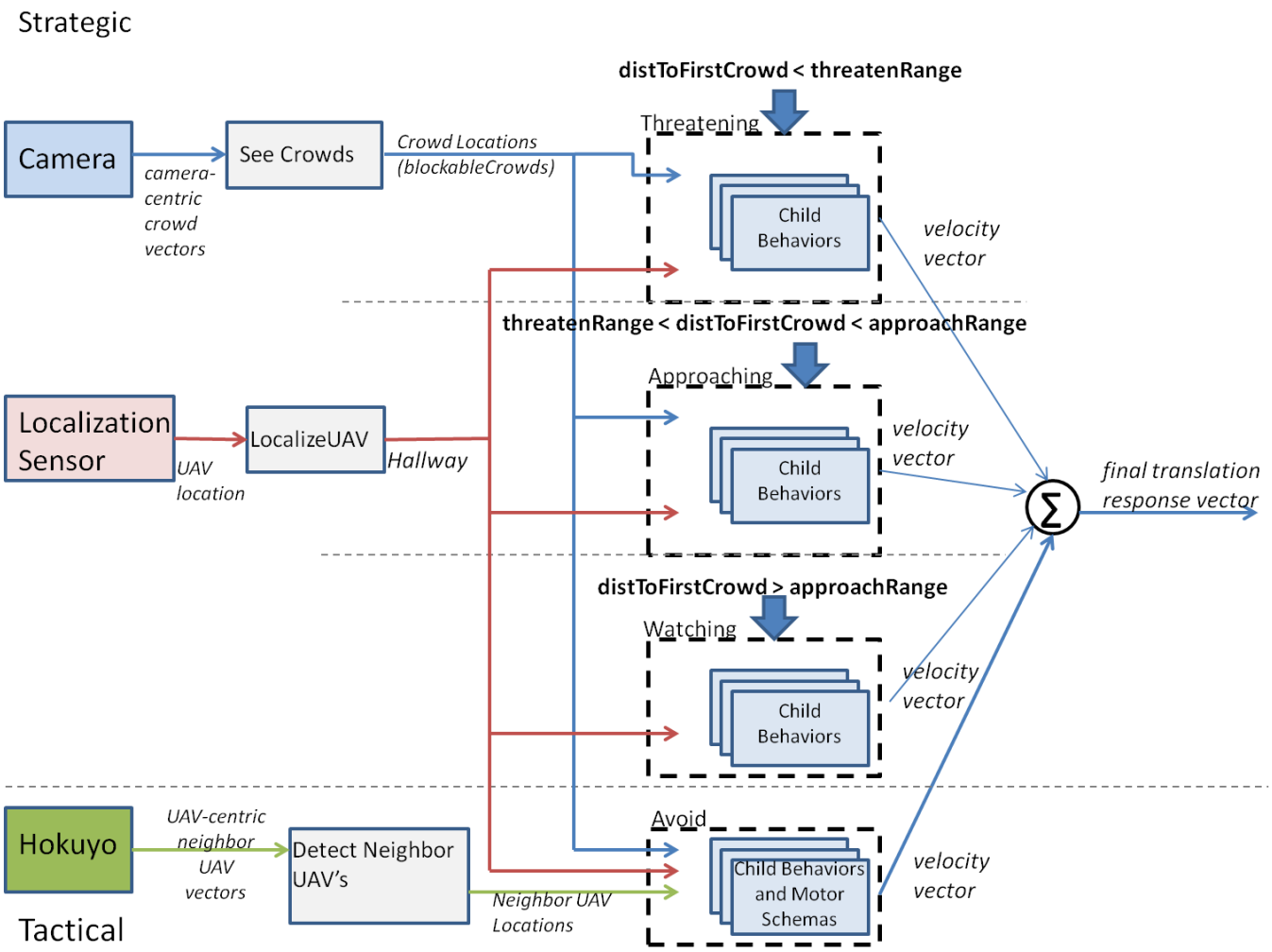
Avoid



As the automation's only tactical behavior, **Avoid** is always active. It is responsible for avoiding collisions with neighboring UAVs, crowds and walls. To do this, it uses four **PerpendicularExponentialIncrease** motor schemas (one for each wall, floor and ceiling), one **AvoidCrowd** child behavior for each currently sensed crowd, and one **AvoidUAV** child behavior for each neighboring UAV.

- **avoidFieldMin_***: these members are translations of the walls, ceiling and floor percepts, in order to provide the minimum-field point for **PerpendicularExponentialIncrease** MS's

Overall Architecture



BlockCrowd2

This is the entry point of the script from Unity. It contains Unity simulation parameters, any initialization necessary, the releasers of the parent behaviors (**Watching**, **Approaching**, **Threatening**), the final sum coordination between the strategic behaviors and tactical avoid behavior, as well as commands to print debugging information to the Unity console.

- **unityScriptAnchor**: Most classes take this as a constructor parameter; it provides them access to percepts and Unity simulation parameters.