VIM-303

Blockly Programming Manual

VISUAL ROBOTICS

For Firmware Release 2.6.0

Scope and other documentation

This manual covers how to program the VIM-303 camera in Blockly. Other relevant manuals include:

- Unboxing and Hardware Assembly Manual
- User Interface Manual
- Settings Manual
- First Picks with VIM-303 Manual

Why Blockly?

Blockly enables programmers of all skill levels to program the VIM-303 system. Graphical programming using Blockly creates a lower barrier to entry for customization of robotic workcells.

Blockly Code Generator

On the VIM-303 camera, Blockly is used as a Python code generator. The Blockly program's blocks are compiled into Python code which is executed by the VIM-303 camera to perform the Programmer's intended behavior. Each block of Blockly code typically compiles into a single Python statement that typically calls to the camera's VIM API (Vision-in-Motion Application Programming Interface).

Toolbox

The **Toolbox** categorizes every Blockly block that the Programmer can use to program the VIM-303 camera (Figure 1). Clicking on a category in the **Toolbox** shows the blocks in that category.

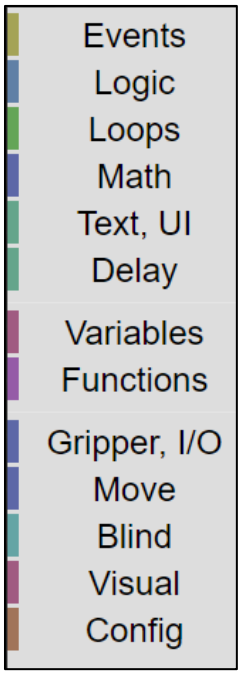

Figure 1 - Blockly Toolbox

Events

The Events category consists of blocks that relate to starting and stopping the program (Figure 2). Figure 3 shows a table of the various event blocks and their function.

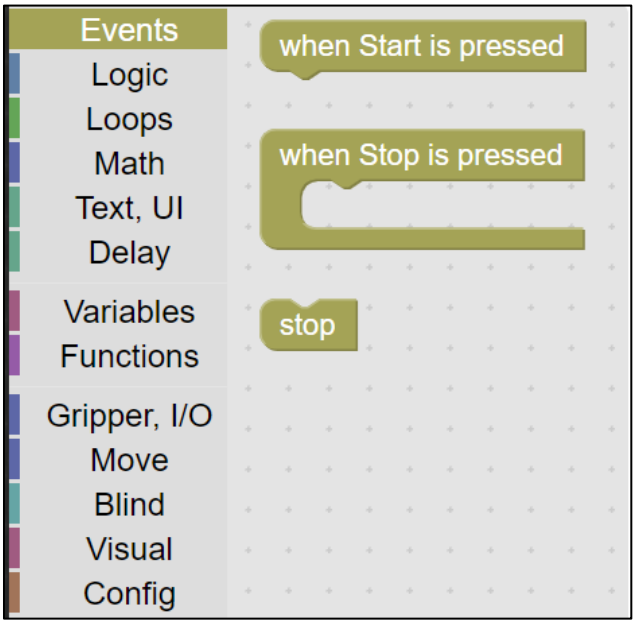

Figure 2 - Events category

Block	Description
when Start is pressed	Place as the first block in the program
when Stop is pressed	Optional block, not required in a program Code to run when stop is pressed (to set robot to a known ending state)
stop	Stops the program after running “when Stop is pressed”

Figure 3 - Event blocks

Logic

The Logic category (Figure 4) consists of an **If** statement and the associated comparison blocks. The **If** statement has a mutator (blue gear) that allows the **If** statement to morph into various forms, including **else if** and **else** clauses (Figure 5). Click the gear to show the mutator menu. Move the desired block piece from the left to the right side of the pop up. Then press the gear to remove the pop up. Figure 6 shows a table of the various logic blocks and their function.

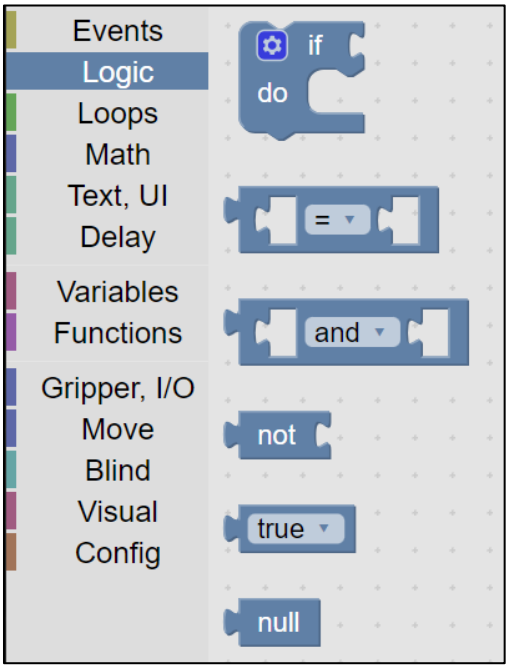

Figure 4 - Logic category

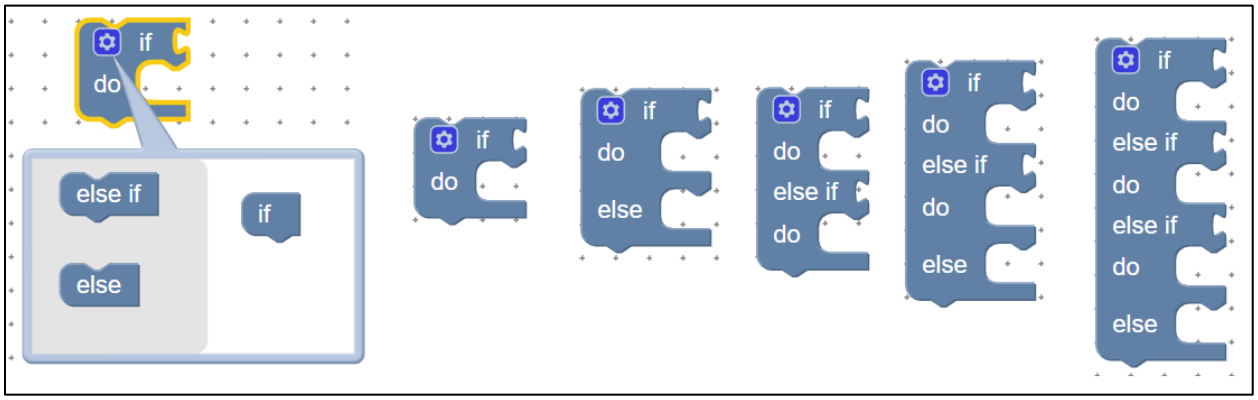

Figure 5 - If statement mutator variants

Block	Description
if	Enables if, else if, else clauses for conditional execution of code
comparison	Compares arguments with equal, not equal, less than, greater than, less than or equal, greater than or equal
not	inverts a logic statement (from true to false or false to true)
true, false	logic constants
null	comparison to something that doesn't exist

Figure 6 - Logic blocks

Loops

The Loops category (Figure 7) consists of various ways to repeat code execution. Figure 8 describes the functionality of the various loop blocks.

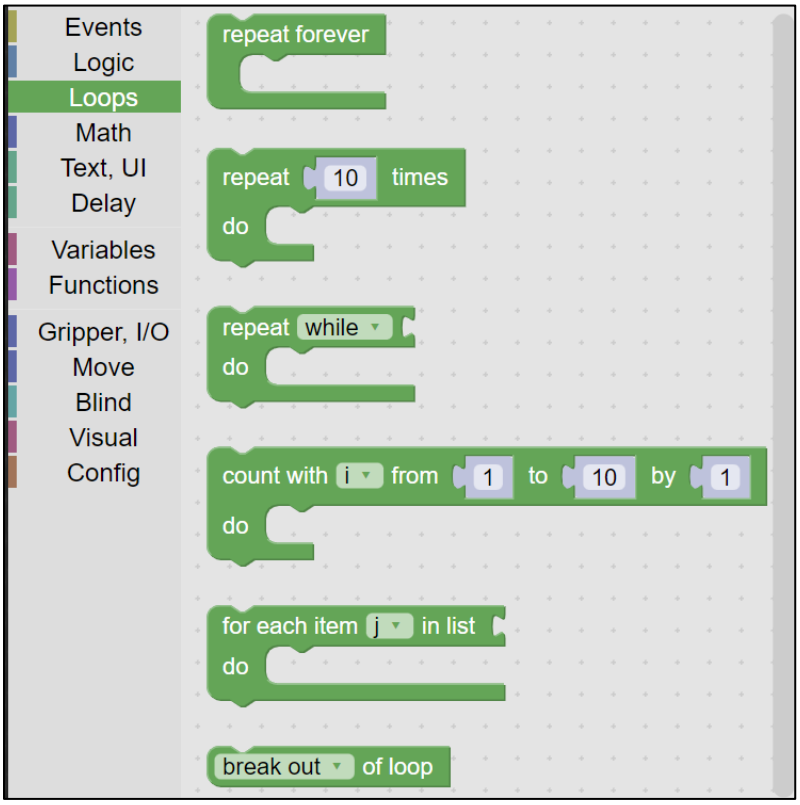

Figure 7 - Loops category

Block	Description
repeat forever	infinite loop (end program by pressing stop)
repeat N times	repeats code N times
repeat while	repeats code while a logic condition is true
count with	a “for” loop that increments a variable each time through loop
for each item in list	a “for” loop that advances through a list
break out of loop	end loop or continue to next iteration

Figure 8 - Loops blocks

Math

The Math category (Figure 9) provides numbers and arithmetic. Figure 10 describes the functionality of the math blocks.

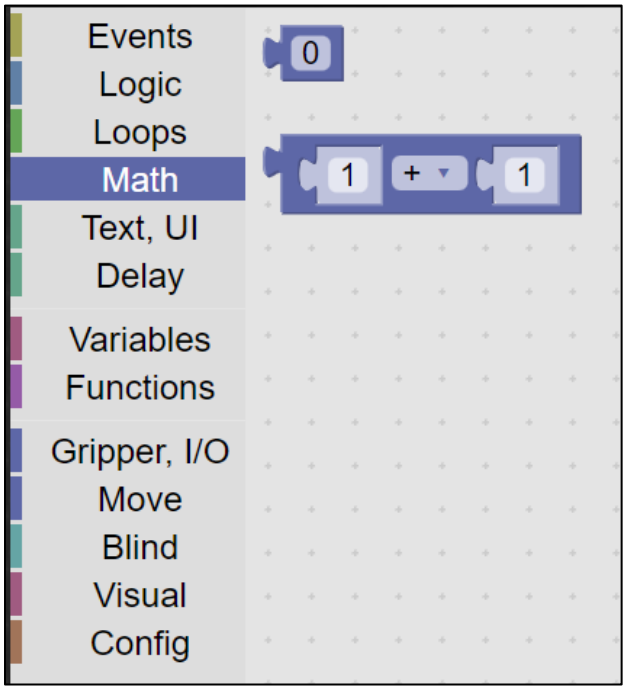

Figure 9 - Math category

Block	Description
number	provides a numerical constant
arithmetic	provides addition, subtraction, multiplication, division, and exponentiation of two numbers

Figure 10 - Math blocks

Text, UI

The Text, UI category (Figure 11) provides debug and code documentation blocks. Figure 12 provides a description of each of the blocks. The UI blocks in v2.54 are primitive (displaying pop-up windows) but provide a method for modifying or regulating program operation by the user.

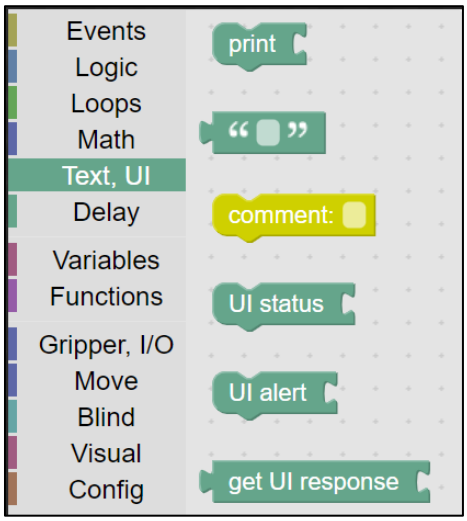

Figure 11 - Text, UI category

Block	Description
print	prints string or variable to Blockly log on Admin page
“ “	text constant (for use in print or comment)
comment	text comment to document code
UI status	generates a pop-up window displaying text. Used for displaying status to the user.
UI alert	generates a pop-up window displaying text that must be closed before program execution continues. Used for requiring the user to perform a task before the robot continues.
get UI response	generates a pop-up window displaying text that requires the user to provide an input string. The input is returned by the block. Used for retrieving input from the user to modify program behavior.

Figure 12 - Text, UI blocks

Delay

The Delay category (Figure 13) provides a single function - a time delay, which is described in detail in Figure 14.

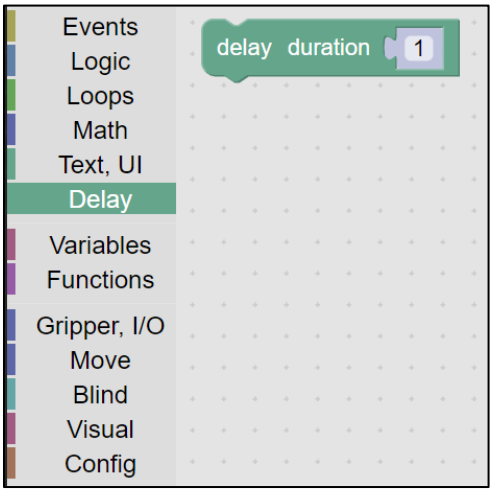

Figure 13 - Delay category

Block	Description
delay	delays a programmable number of seconds

Figure 14 - Delay block

Variables

The Variables category (Figure 15) provides numeric, array, boolean, or string variables that can be set and read. Figure 16 describes the blocks in detail.

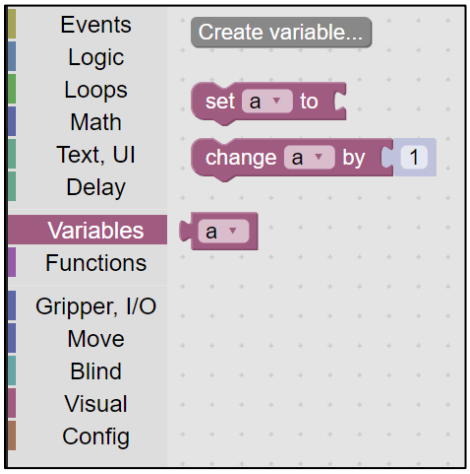

Figure 15 - Variable category

Block	Description
create variable	create a new variable (places a set of 3 functions into toolbox category)
set	sets a variable to a value
change	increments a variable
get	gets the current value of a variable

Figure 16 - Variable blocks

Functions

The Functions category allows the Programmer to create subroutines and execute them (Figure 17). The Function block has a mutator to allow variables to be passed. When a new function is created, new blocks are created that allow you to call the function (Figure 18). For example, for the function “do something” the code for the function is in the block “to do something” while calling the function is “do something”. Figure 19 lists each of the blocks in the function category.

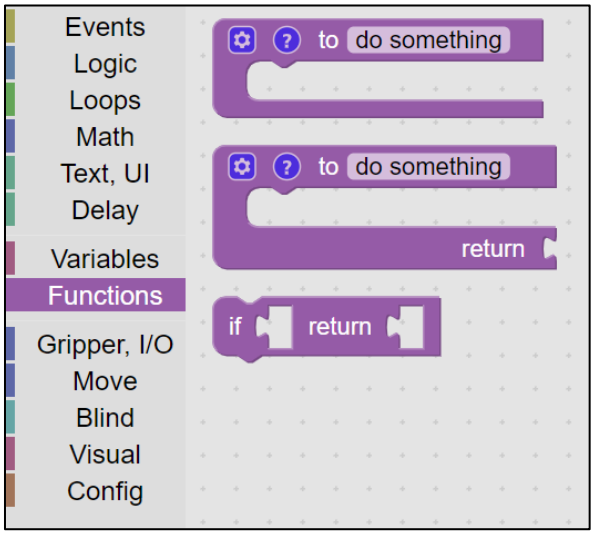

Figure 17 - Function category

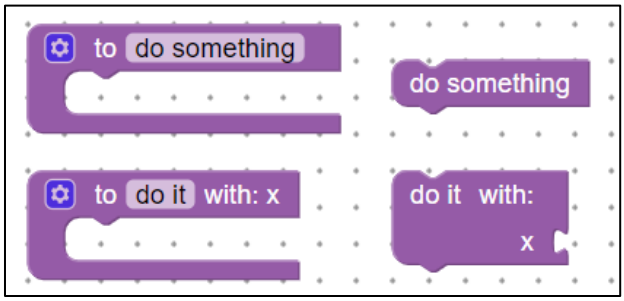

Figure 18 - Defining and calling functions

Block	Description
to do something	create a subroutine
to do something, return	create a subroutine (function) that returns something
if, return	conditional return statement
do something	execute function (“do something” is replaced with function name)

Figure 19 - Function blocks

Gripper, I/O

The Gripper, I/O category combines I/O functions supported by the robot controller, including actuation of grippers, digital input, and digital output (Figure 20). Macro blocks which wait for an input to change and to set an output for a certain period of time, simplify code development and readability. The list of Gripper blocks is shown in Figure 21.

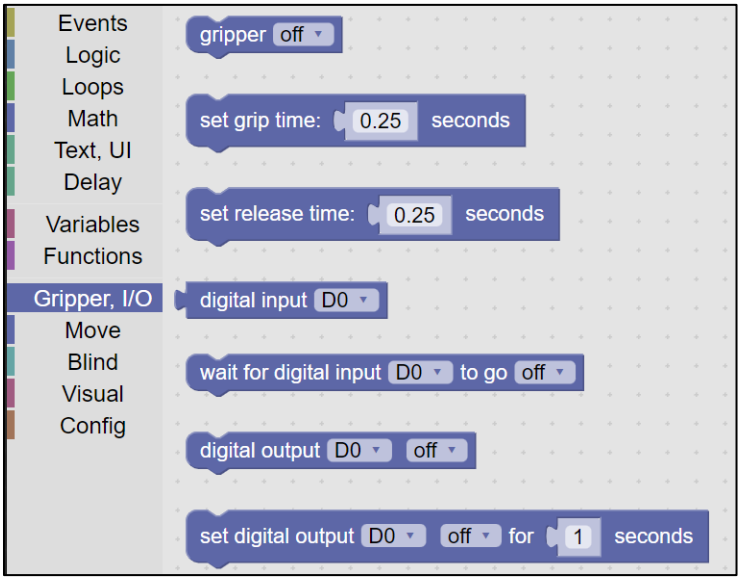

Figure 20 - Gripper category

Block	Description
gripper	turns gripper on/off (currently Digital Output 0)
digital input	reads state of a digital input
wait for digital input	waits until digital input achieves desired state (often used to synchronize with other equipment)
digital output	set state of digital output
set digital output for duration	set state of digital output for a certain duration (often used to indicate a state to other equipment)

Figure 21 - Gripper blocks

Move

The Move category provides blind moves, adjusts robot speed, and provides a list of waypoints created from the Movement Editor (Figure 22). Moves by default move to “nowhere”, which does nothing. When a move block is placed in the canvas, the desired waypoint is selected from the pulldown menu. The list of all of the move blocks is described in Figure 23.

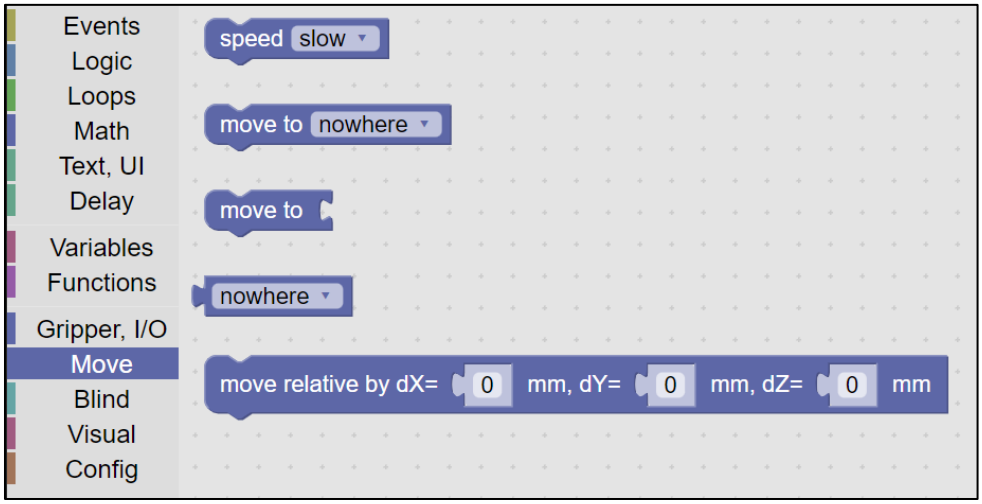

Figure 22 - Move category

Block	Description
speed	sets robot speed from a pulldown list
move to waypoint	moves to a named waypoint
move to variable waypoint	moves to a variable that contains a named waypoint
waypoint	pulldown list to select a waypoint
move relative	move from current location by X,Y,Z offset

Figure 23 - Move blocks

Blind

The Blind category provides blind macro blocks for pick and place (Figure 24). Detailed descriptions of the blind blocks is shown in Figure 25. The blind blocks move to specified locations, defined by waypoints, rather than using vision to guide the destination.

Blind pick assumes the top of the object is located at the specified waypoint. The robot will move above this location, per the retract distance (programmed with a config block), move down to touch the object, actuate the gripper, and lastly, retract.

Blind place moves above the placement location per the retract distance, moves down to the specified waypoint, releases the gripper, and lastly retracts.

Blind stack is used to repeatedly place objects one upon another. The index specifies the height to place the object. The first time the blind stack is called, use index = 1 and it will place at the specified waypoint. Calling blind stack with an index greater than 1 will place at the specified waypoint offset in Z by the stack height per object.

Blind palletize provides a high degree of flexibility in placing objects in a grid pattern, described in Figure 26.

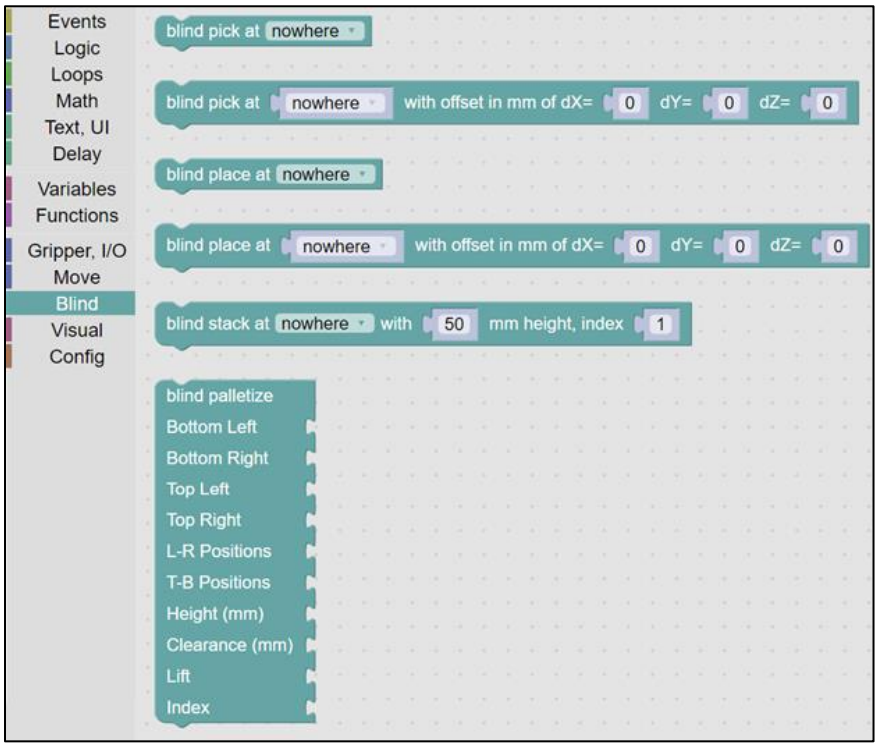

Figure 24 - Blind category

Block	Description
blind pick	picks object who's top-center is located at waypoint moves to top-center, engages gripper, and retracts
blind pick with offset	blind pick of a variable waypoint with offset from top-center
blind place	blind place object's top-center at waypoint moves to top-center + retract, moves to top-center, disengages gripper
blind place with offset	blind place to variable waypoint with offset from top-center
blind stack	stack top-center of object at waypoint offset placement location by height per index
blind palletize	blind place in a pallet (see Figure 26)

Figure 25 - Blind blocks

Blind Palletize

The **Blind Palletize** block provides a high degree of placement flexibility. Figure 26 shows the five different configurations for this block, while Figure 27 shows the use of the block on the canvas for the various configurations. For each configuration, waypoints are used to specify the corner locations of the pallet by indicating the top-center locations of the object to be palletized. Up to four waypoints can be specified, to fully define the pallet geometry (A), which can have a rectangular, parallelogram, or trapezoidal shape. These waypoints are the **Bottom Left**, **Bottom Right**, **Top Left**, and **Top Right**. The blockly examples (Figure 27) show waypoint definitions (from the Move category) attached to the waypoint sections of the palletize block. Fewer than four waypoints can be specified, in order to create simpler palletization scenarios. For example, if the **Top Right** waypoint is unspecified, this location is extrapolated from the three specified points (**Bottom Left**, **Bottom Right**, and **Top Left**) as shown in example (B). Linear pallets can be specified using just two waypoints, using the **Bottom Left** and **Bottom Right**, shown in example (C), or using the **Bottom Left** and **Top Left**, shown in example (D). If only the **Bottom Left** waypoint is specified, the palletize block will simply stack at the **Bottom Left** waypoint, shown in example (E).

The number of grid positions between and including the **Bottom Left** and **Bottom Right** is specified by the **L-R Positions** parameter. Similarly, the number of grid positions between and including the **Bottom Left** and **Top Left** is specified by the **T-B Positions** parameter.

The **Height** parameter works the same way as it does for the **Blind Stack** block, which specifies the height between layers of the pallet, which is typically the height of the object being palletized.

The **Clearance** and **Lift** parameters are illustrated in Figure 28. When performing blind palletization, it is helpful for the robot to bring the object close to its destination and then snug it up to other objects when close. The **Clearance** parameter specifies the lateral (X,Y) offset (in mm) for initial placement of the object before moving to the final position. The **Lift** parameter specifies the vertical (Z) offset (in mm) from the final position when performing the initial placement. When palletizing an object with nonzero **Clearance** and **Lift** parameters specified, the robot will move the object to the retract distance above the placement location, considering the clearance. It will then lower the object to the **Lift** distance above the final placement location, offset in X and Y by the **Clearance** distance. It will then move to the correct (X, Y) lateral position, still offset in height (Z) by the **Lift** distance, and then will finally lower the object to the final location. This will have the effect of close-packing boxes, for example.

The last parameter, **Index**, is a variable that starts at 1 to indicate the object should be set at the **Bottom Left** location. Incrementing **Index** will cause the palletize block to compute positions first along the **L-R** direction between the **Bottom Left** and **Bottom Right** and then increment along the **T-B** direction between the **Bottom Left** and **Top Left** and then increment in **Height**. The numbers in Figure 26 indicate the palletizing order for the first layer.

The names bottom, top, left, and right, are arbitrary. They specify the order that the palletizing algorithm will compute placement locations. Waypoints can be defined for a different purpose, but it may be useful for the Programmer to orient themselves when specifying waypoints so that the Bottom Left of the pallet is oriented accordingly.

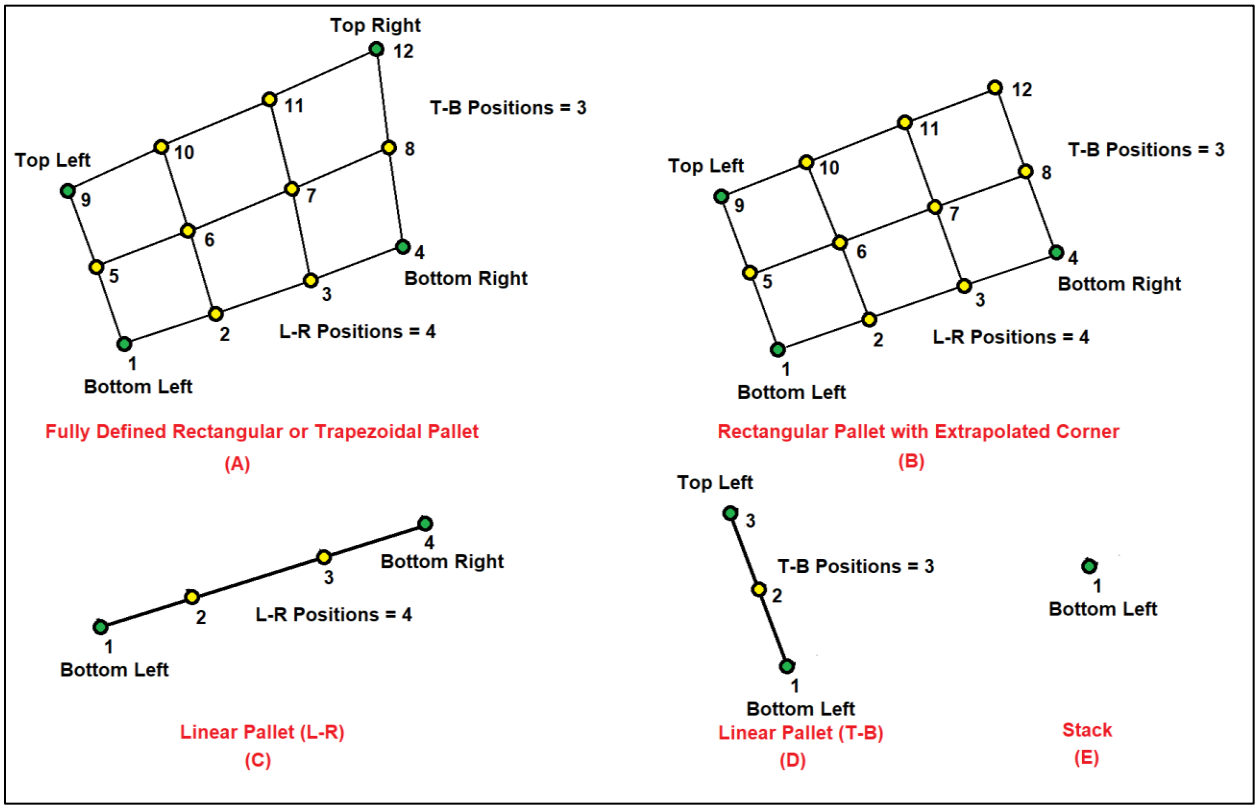

Figure 26 - Pallet Configurations

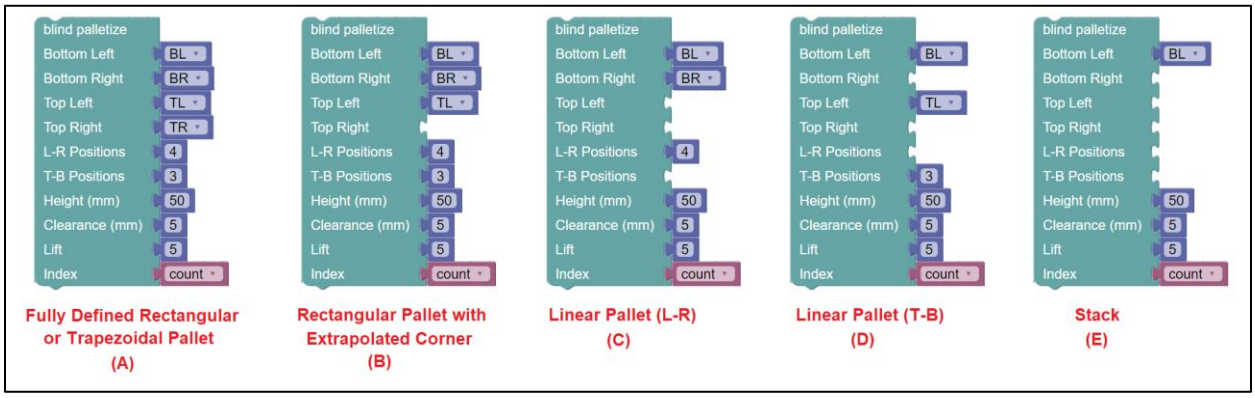

Figure 27 - Pallet block usage

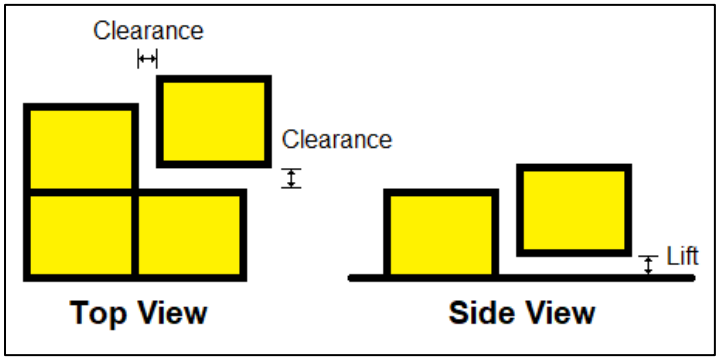

Figure 28 - Clearance and Lift definition

Visual

The Visual category includes blocks to visually pick, observe, and identify objects (Figure 29). A list of all of the visual blocks is shown in Figure 30. One of VIM-303's outstanding features is the ability to visually pick an object by name. The **visual pick** block allows the Programmer to specify the object to be picked from a pulldown list (Figure 31). The **visual pick variable** block allows the Programmer to pick an object that is specified by a variable loaded with a string representing the object, enabling complex run-time behavior (Figure 31). The **create list** block allows multiple objects (either variables or the list block) to be connected to the **visual pick variable** block. Figure 32 shows how multiple different objects can be picked, and custom behavior performed depending on which object was picked. Figure 33 shows how the status of the visual pick can be used to handle error conditions.

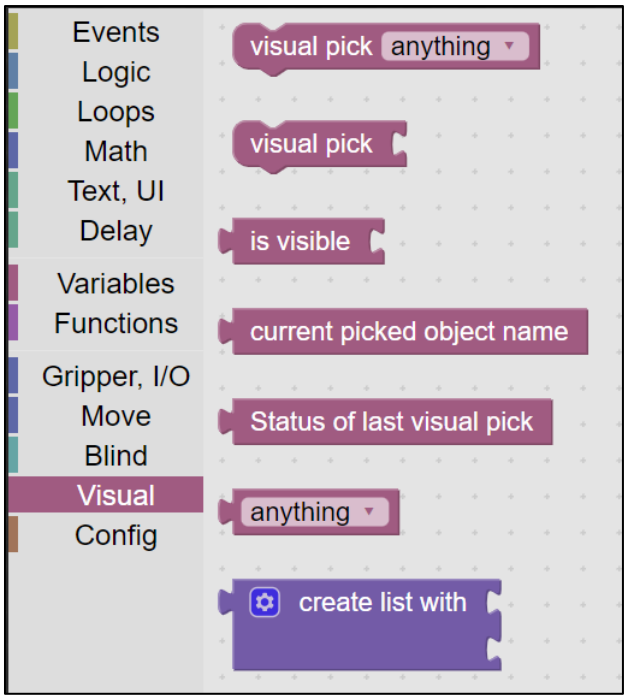

Figure 29 - Visual category

Block	Description
visual pick object	visually pick an object from the object list
visual pick variable object	visually pick an object listed in a variable
is object visible	return true if an object(s) is visible
current picked object name	return the name of the object that was picked
status of last visual pick	returns success, object_lost, object_out_of_bounds, canceled, error, ignored, timeout, or unknown
list of object names	provide the name of an object (for a variable or a comparison)
create a list (of objects)	create a list (used for selecting multiple objects)

Figure 30 - Visual blocks

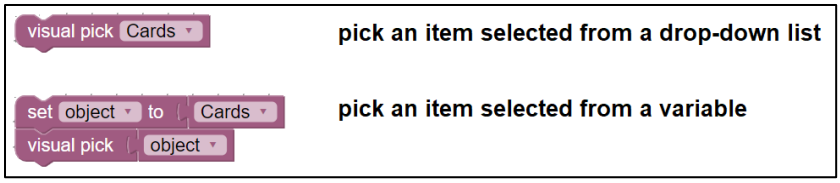

Figure 31 - Visual pick examples

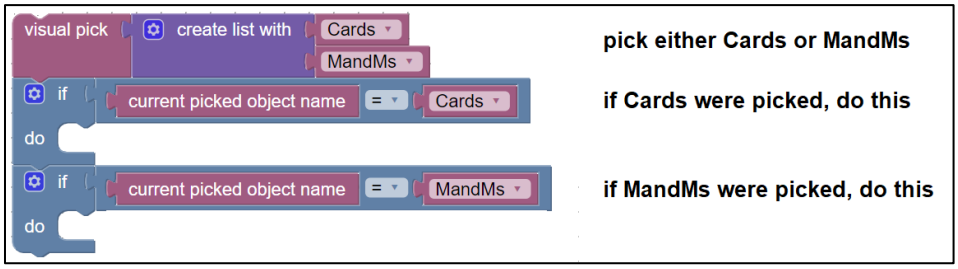

Figure 32 - Visual pick of multiple items

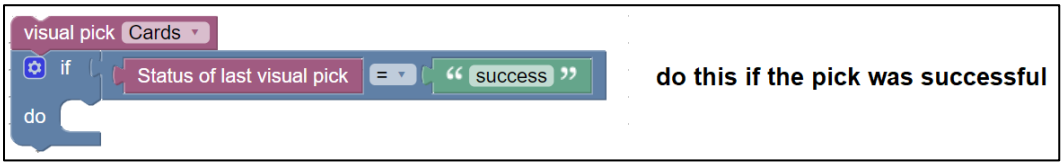

Figure 33 - Error checking of visual picking

Config

The Config category provides the ability to change various system settings (Figure 34). Figure 35 tabulates the various types of configuration items that can be set in a Blockly program. Settings can also be set in the **Settings** tab.

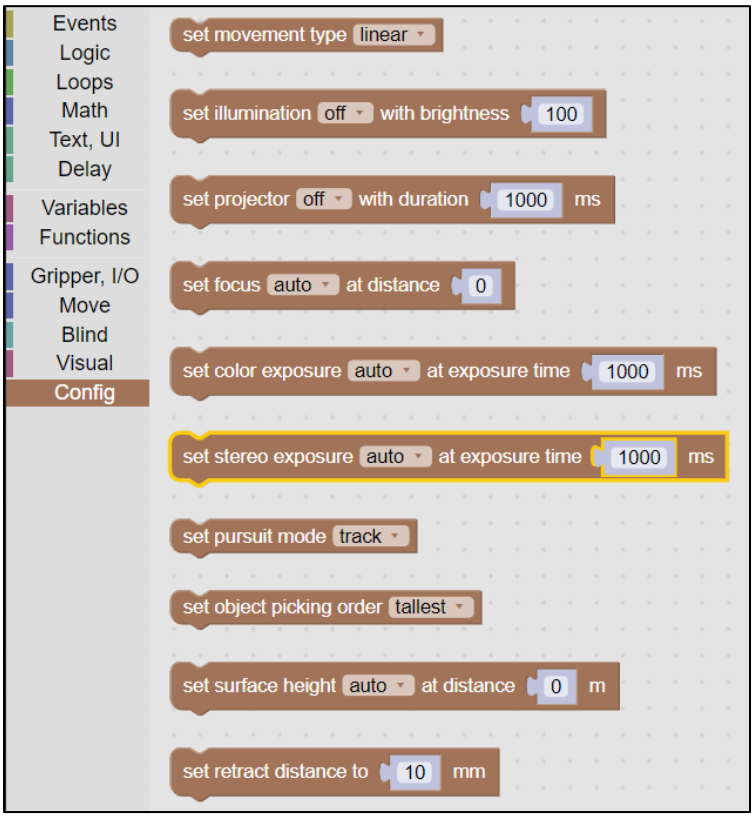

Figure 34 - Config category

Block	Description
movement type	sets linear or joint moves for waypoint moves and blind pick and place
illumination	turn illumination LEDs on/off and set brightness
projector	turn IR projector for stereo camera on/off
focus	set color camera focus to auto or manual
color exposure	set exposure of color camera to auto or manual
stereo exposure	set exposure of stereo camera to auto or manual
pursuit mode	configures the way picking in motion occurs track = robot moves camera over object before picking blind = robot picks object as soon as it is seen
object picking order	configures the way objects are selected tallest = tallest object in field of view newest = most recent object seen
surface height	sets manual or automatic surface height
retract distance	configures the retract distance for pick and place

Figure 35 - Config blocks

Sample Programs

The simplest example of a visual pick and blind place is shown in Figure 36. When the **Start** button is pressed, the robot moves to the waypoint PickZone. It visually picks the object Cards, whether it be statically within the field of view or if it is moving, such as on a conveyor. Once the object has been picked, the robot moves to the waypoint PlaceZone and sets the object down.

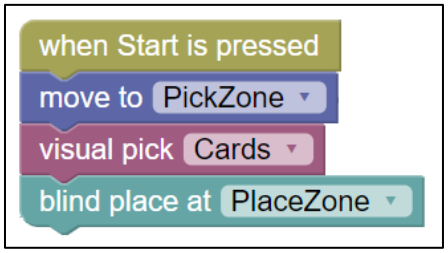

Figure 36 - Simple visual pick and blind place

Figure 37 shows a palletizing example. Six cards are picked from a conveyor and palletized on a grid of 3x2. The variable Cards is used to index through the pallet.

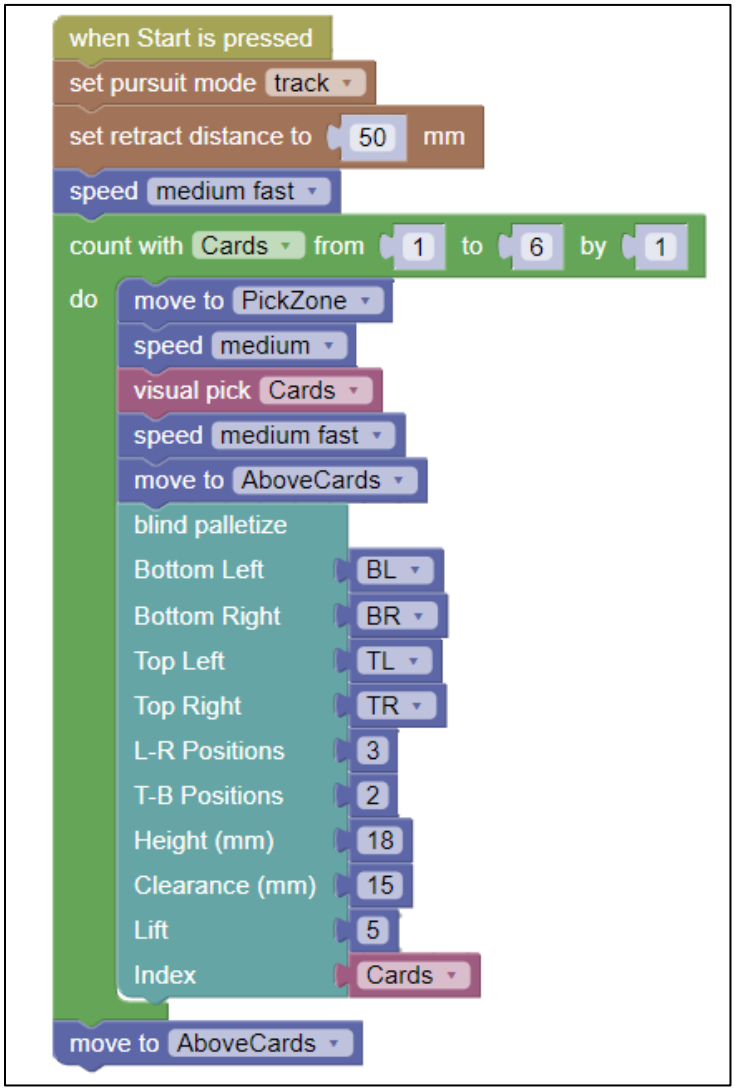

Figure 37 - Palletizing example

Figure 38 shows a complex visual picking example. Cards and M&Ms are picked from a conveyor and sorted into two different locations. The beginning of the program defines the surface height of the conveyor, sets the pursuit mode to Track for the best picking accuracy, and sets the retract distance to ensure that taller objects are placed properly. Two variables, Cards and MMs, are used to keep track of the state of the palletizing and stacking. Visual picking is done with a list of two objects, Cards and MandMs. Depending on the object that was picked, one of two functions, PalletizeCards and StackMMs, is called. The PalletizeCards function increments the Cards variable and performs a palletization of the cards on a 3x2 grid. The StackMMs function increments the MMs variable and performs a stacking of the M&Ms.

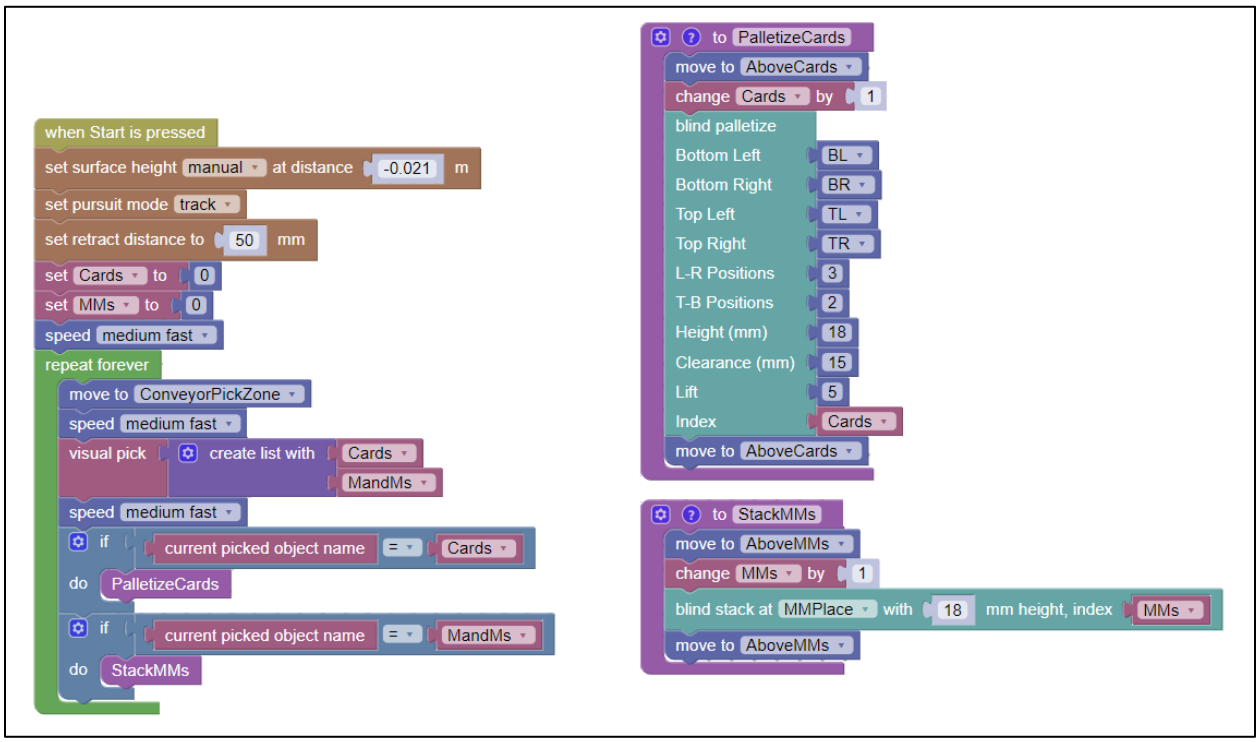

Figure 38 - Visual sorting example