USER'S MANUAL

TABLE OF CONTENTS:

1- Getting started	2
1.1- Booting the system	2
2- Description of each PySection	3-38
2.01- General Purpose Hand A	3-4
2.02- Pipetting Hand – 0.2 to 2.0 ml	5-6
2.03- Internal Standard Hand – 0.010 to 0.200	7-8
2.04- Rack GC Vial (Rack 1)	9-10
2.05- Rack 13 x 100 (Rack 2)	11-12
2.06- Rack 16 x 100 (Rack F)	13-14
2.07- Weighing - Liquid Transfer – 16 x 100 mm Test Tubes	15-16
2.08- Capping – 16 x 100 mm tubes	17-18
2.09- Dilute and Dissolve – 16 x 100 mm Tubes (Vortex)	19-20
2.10- Evaporation	21-22
2.11- Tumble Mixer – 16 x 100 mm Tube	23-24
2.12- Liquid/Liquid Extraction	25-26
2.13- Crimp capping	27-28
2.14- Test Tube Dispenser – 16 x 100 mm test tubes	29-30
2.15- Disposal	31-32
2.16- Centrifuge - 16 x 100 mm Tubes	33-34
2.17- Power and Event Controller (PEC)	35-36
2.18- Zymate II Core System	37-38
3- Advanced operations	39
3.1- Re-teaching positions	39
4- Loop, conditional & pausing commands	40-47
4.1- DO / ENDDO	40-41
4.2- GOTO	42-43
4.3- IF / THEN	44-45
4.4- TIMER (x) / WAIT FOR TIMER x	46-47

1 – Getting started

✐ **1.1- Purpose:** Booting the system.

1. Insert the Boot Disk in the Controller.
2. Turn ON the power of the Controller, the Robot, the PyStations and the PC.
3. In the main menu (PC) select System V On-line and press Enter.

💣 If apperars an error message ("Controller not responding"), don't panic, it's normal. Just press the Enter button in the computer to continue.

4. Now it's loading the system to the Controller. After a while, we'll access to the Zymark's main menu.

💣 Now it's time to load the dictionary.

5. In the main menu, go to System and then Load dictionary.
6. Select the proper dictionary (JO-CHAN.ZYD in our case) and press Enter.

💣 To avoid getting an error message, remove the hand from the robot (if any)

💣 After a while, we'll have the system propery configured and loaded.

💣 The Original Dictionary File was 0393AP.ZYD but we used our modified one (JO-CHAN.ZYD).

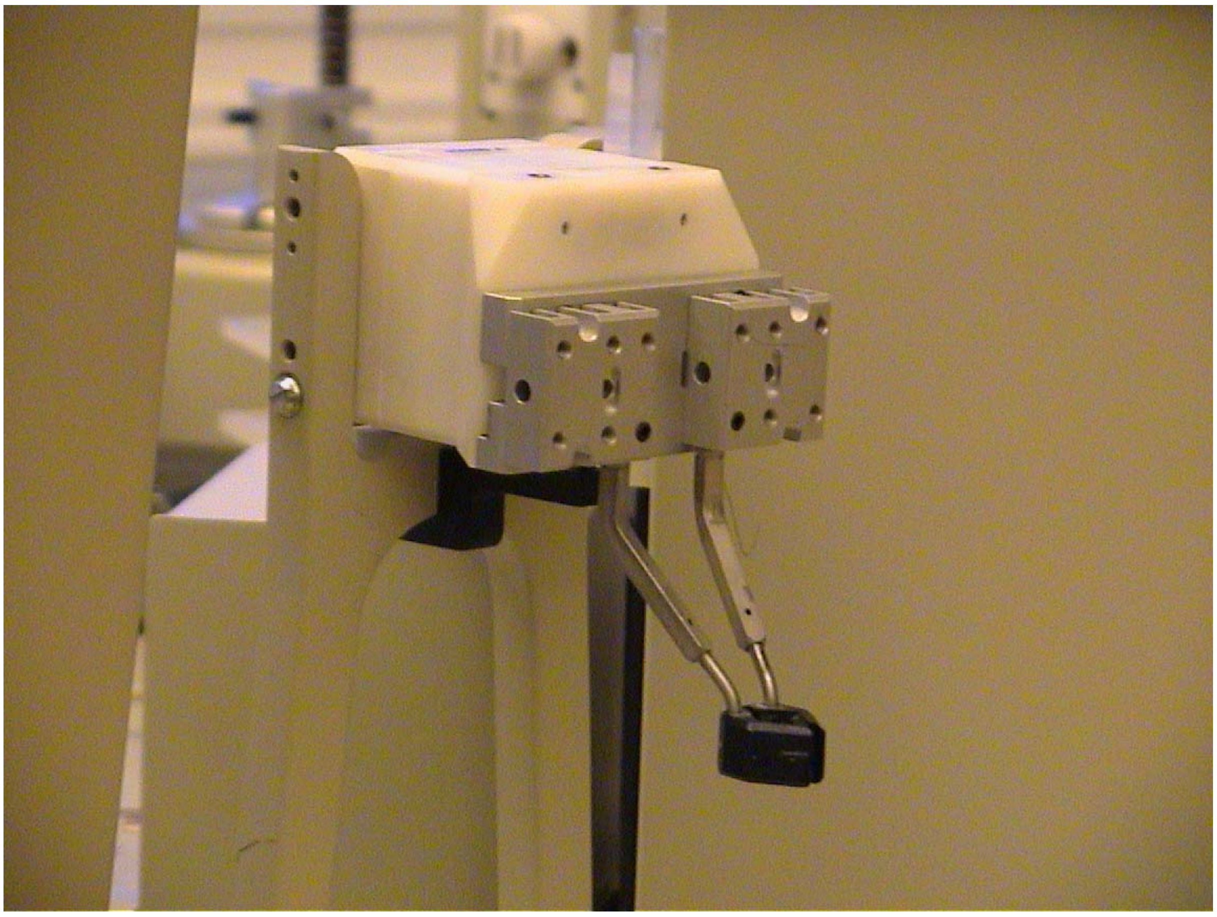
💣 The variable AIR.CONFIRM.SENSOR must be 0 in order to use air pressure.

💣 To test each command or instruction, Manual Control was used (in the main menu, go to Methods and then Manual Control).

💣 To avoid checkings and error messages, change ERROR.CHECK to value 0.

2 - Description of each PySection

✐ **2.01- Purpose:** General Purpose Hand A.



2 - Description of each PySection

Description:

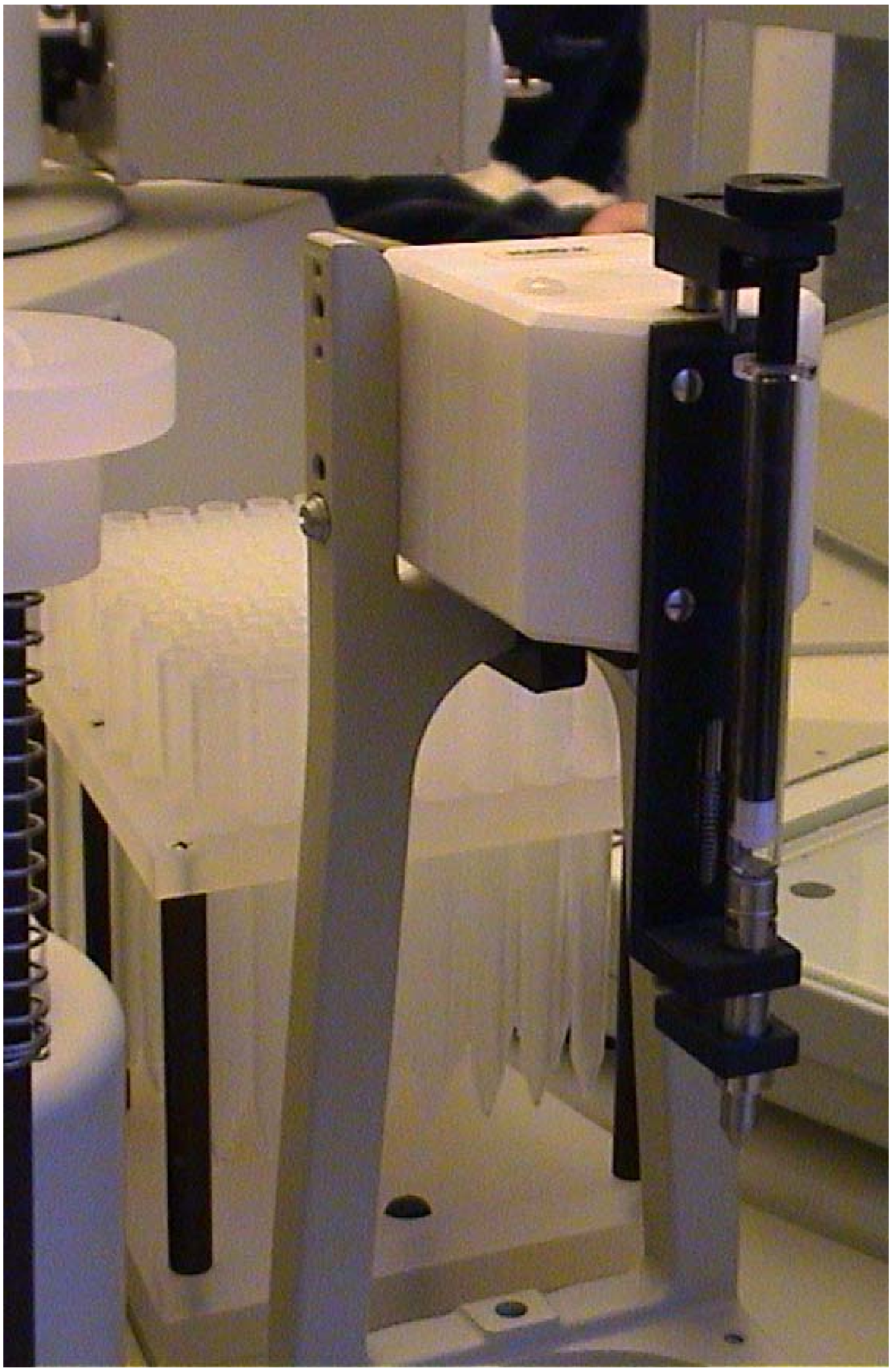
The General Purpose Hand PySection provides capability to grasp and move containers ranging in size from 9 to 16 mm from station to station around the benchtop. This hand can also be used with Capping Station PySection for screw-capping operations involving 9 to 16 mm capped containers. The hand comes equiped with optional grip-release wires on the finger pads for use with 11 mm vials. These wires must be removed when using the hand for screw-capping. If both 11 mm vial manipulation and capping are to be performed, there must be two "HAND.A's" in the system.

Available commands:	GET.HAND.A PARK.HAND
Variables:	(none)
Testing Program:	JOOCHAN.HAND.A

- display off
- get.hand.a
- prompt Inster source position (Rack 1): 1 to 50
- input rack.1.index
- get.from.rack.1
- prompt Insert evaporator position: 1 to 6
- input evaporator.index
- put.into.evaporator
- get.from.evaporator
- prompt Insert destination position (Rack 1): 1 to 50
- input rack.1.index
- put.into.rack.1
- park.hand

2 - Description of each PySection

✐ **2.02- Purpose:** Pipetting Hand – 0.2 to 2.0 ml.



2 - Description of each PySection

Description:

This PySection can quantitatively transfer 0.2 to 2.0 ml of liquid using 2 ml disposable pipet tips between user-designated samples sources and destinations. This PySection includes a Syringe hand with 2.5 ml syringe, automatic tip ejector, parking station, rack to hold 105 pipet tips and a bag of 250, 2 ml disposable pipet tips. An Auxiliary Rack PySection that accomodates an additional 105, 1 ml pipet tips can also be supported by this PySection. It is recommended that a Disposal PySection be included in the PySection for convenient disposal of used tips.

Available commands:	GET.HAND.K GET.2ML.TIP ASPIRATE.2ML.TIP DISPENSE.2ML.TIP PARK.HAND

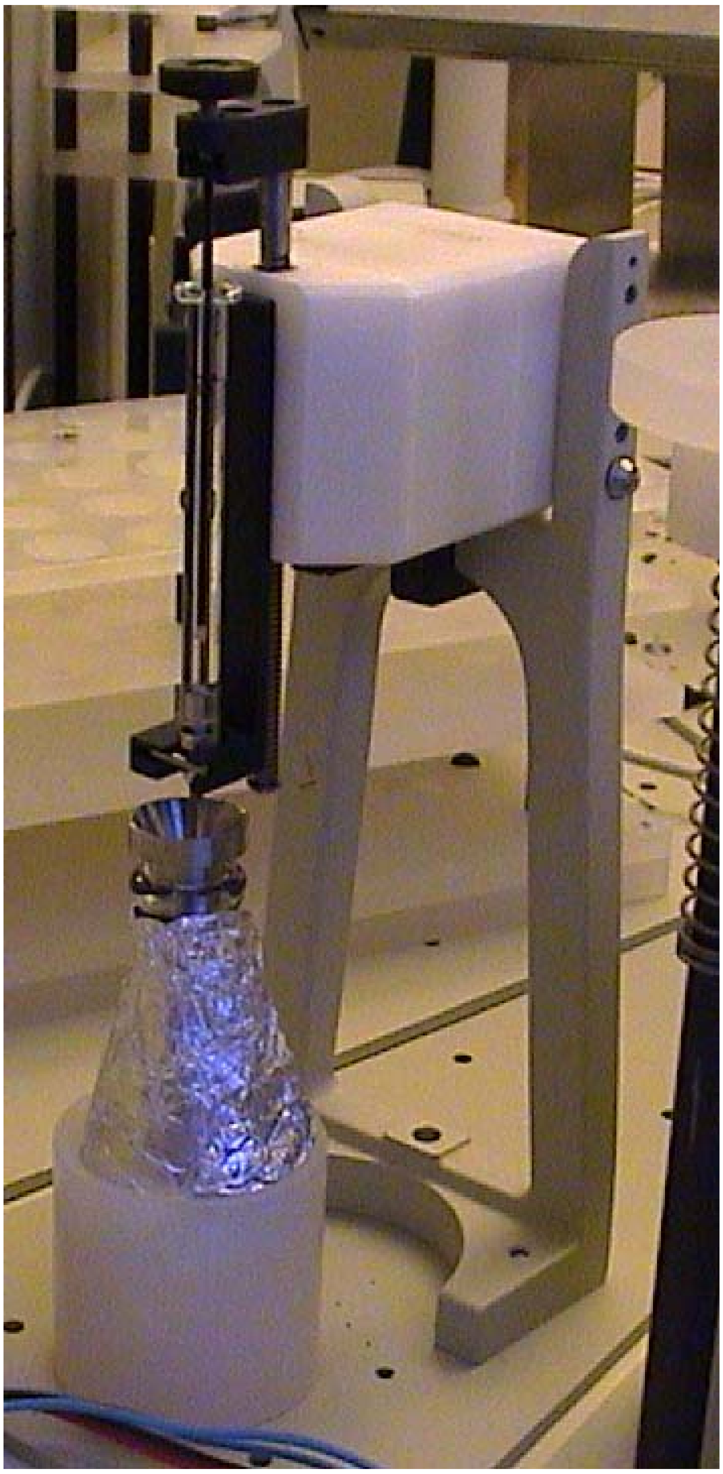
Variables:	PIPET.VOLUME = 0.2 to 2.0 (ml)
	NUMBER.OF.2ML.PREWETS = 0 to X
	ASPIRATE.2ML.PAUSE.TIME = 0 to X (sec)
	DISPENSE.2ML.PAUSE.TIME = 0 to X (sec)

Testing Program:	JOOCHAN.HAND.K

- display off
- get.hand.k
- get.2ml.tip
- prompt Inster source position (Rack 2): 1 to 50
- input rack.2.index
- move.over.rack.2
- prompt Insert aspiration volume: 0.2 to 2.0
- input pipet.volume
- aspirate.2ml.tip
- prompt Insert destination position (Rack 1): 1 to 50
- input rack.1.index
- move.over.rack.1
- dispense.2ml.tip
- dispose.to.waste
- park.hand

2 - Description of each PySection

✐ **2.03- Purpose:** Internal Standard Hand – 0.010 to 0.200.



2 - Description of each PySection

Description:

This PySection can quantitatively transfer 0.010 to 0.200 ml of liquid using a 250 µl syringe between the internal standard source and the sample. The PySection includes a Syringe Hand with a 250 µl syringe, a reusable dripping-lock needle, and a covered reservoir for the internal standard.

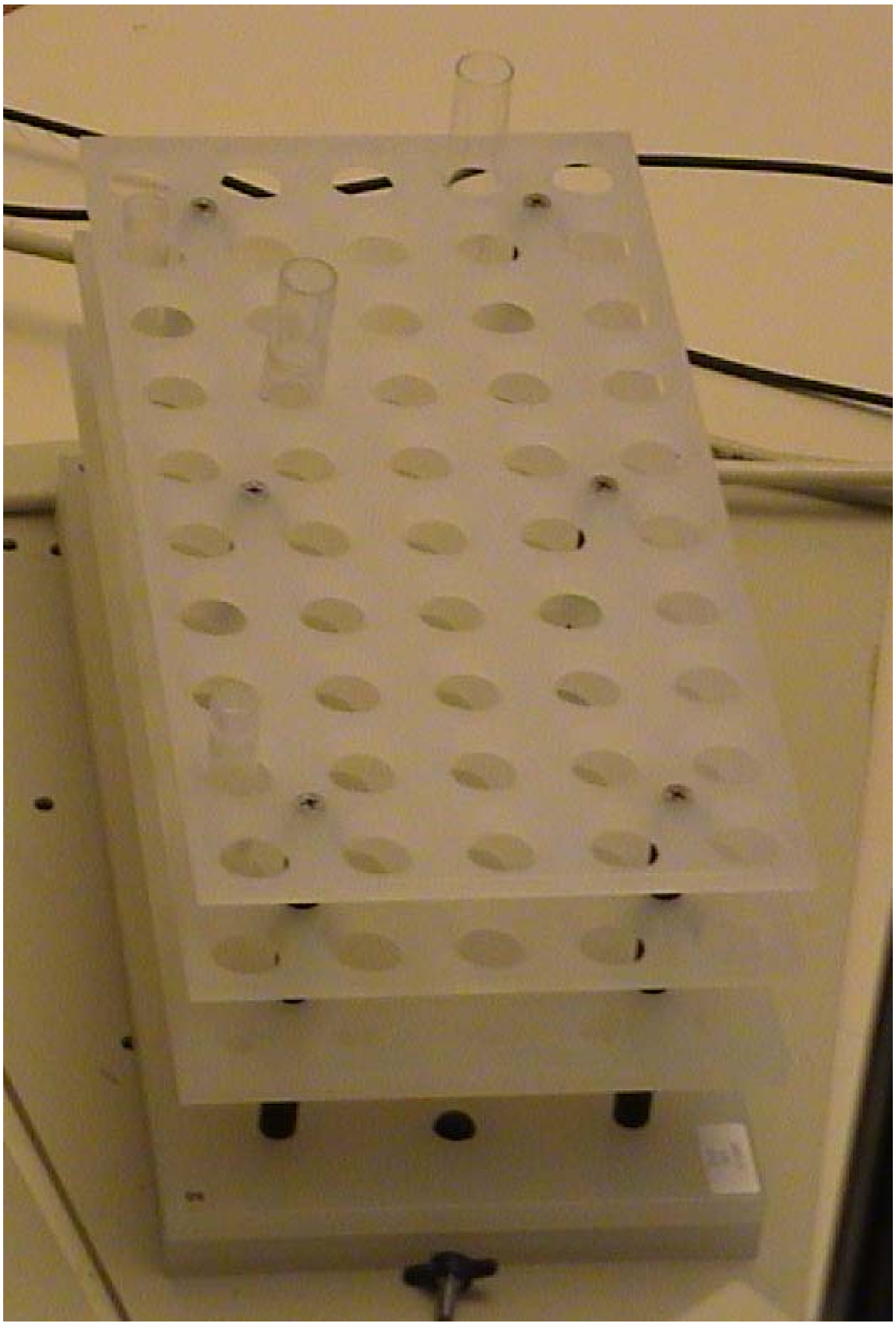
NOTE: Before executing DISPENSE.ISTD, MOVE.OVER.xxxxx is required.

Available commands:	GET.ISTD.HAND ASPIRATE.ISTD DISPENSE.ISTD PARK.HAND
Variables:	ISTD.VOL = 0.01 to 0.24 (ml)
Testing Program:	JOOCHAN.HAND.ISTD

- display off
- get.istd.hand
- prompt Insert aspiration volume: 0.01 to 0.2
- input istd.vol
- aspirate.istd
- prompt Insert destination tube position (Rack 2): 1 to 50
- input rack.2.index
- move.over.rack.2
- dispense.istd
- park.hand

2 - Description of each PySection

✐ **2.04- Purpose:** Rack GC Vial (Rack 1).



2 - Description of each PySection

Description:

This Rack PySection provides storage for 11 mm GC vials. Each container in the rack has an associated volume and capped status that is automatically updated as the container is moved around the benchtop. The rack is made by polypropylene and is mounted on a snap-base for easy removal and replacement. Containers are held upright in the rack with adecuate spacing to permit individual container access with Zymate Hands.

Available commands:	GET.FROM.RACK.1 PUT.INTO.RACK.1 MOVE.OVER.RACK.1

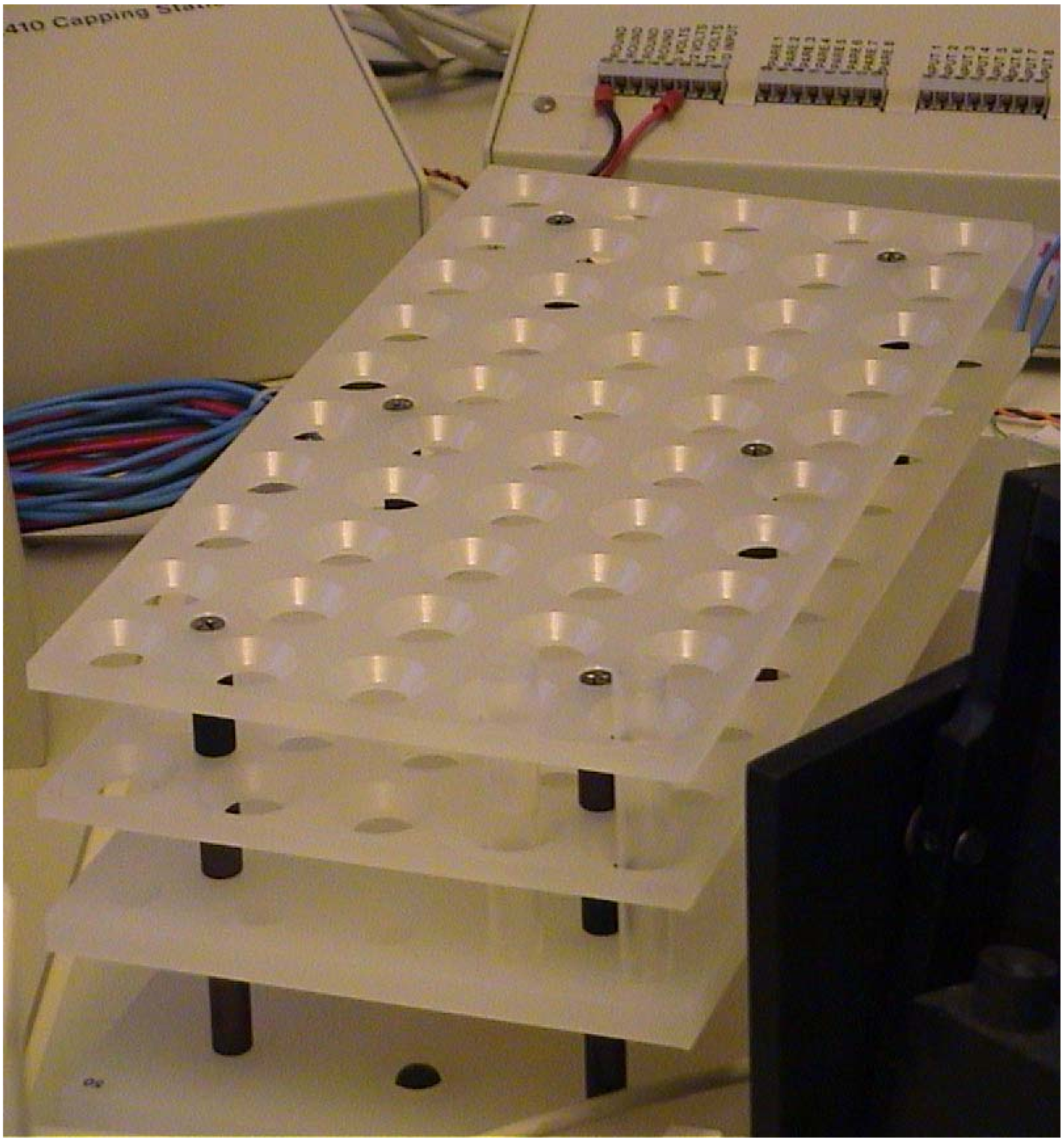
Variables:	RACK.1.INDEX = 1 to 50
	INITIAL.VOLUME.RACK.1 = 0 to 12 (ml)
	CONTAINER.CAPPED.RACK.1 = YES or NO

Testing Program:	JOOCHAN.RACK.1

- display off
- get.hand.a
- prompt Inster source position (Rack 1): 1 to 50
- input rack.1.index
- get.from.rack.1
- prompt Insert destination position (Rack 1): 1 to 50
- input rack.1.index
- put.into.rack.1
- park.hand

2 - Description of each PySection

✐ **2.05- Purpose:** Rack 13 x 100 (Rack 2).



2 - Description of each PySection

Description:

This Rack PySection provides storage for 13 x 100 mm test tubes. Each container in the rack has an associated volume and capped status that is automaticaly updated as the container is moved around the benchtop. The rack is made of polypropylene and is mounted on a snap-base for easy removal and replacement. Containers are held upright in the rack with adecuate spacing to permit individual container access with Zymate Hands.

Available commands:	GET.FROM.RACK.2 PUT.INTO.RACK.2 MOVE.OVER.RACK.2

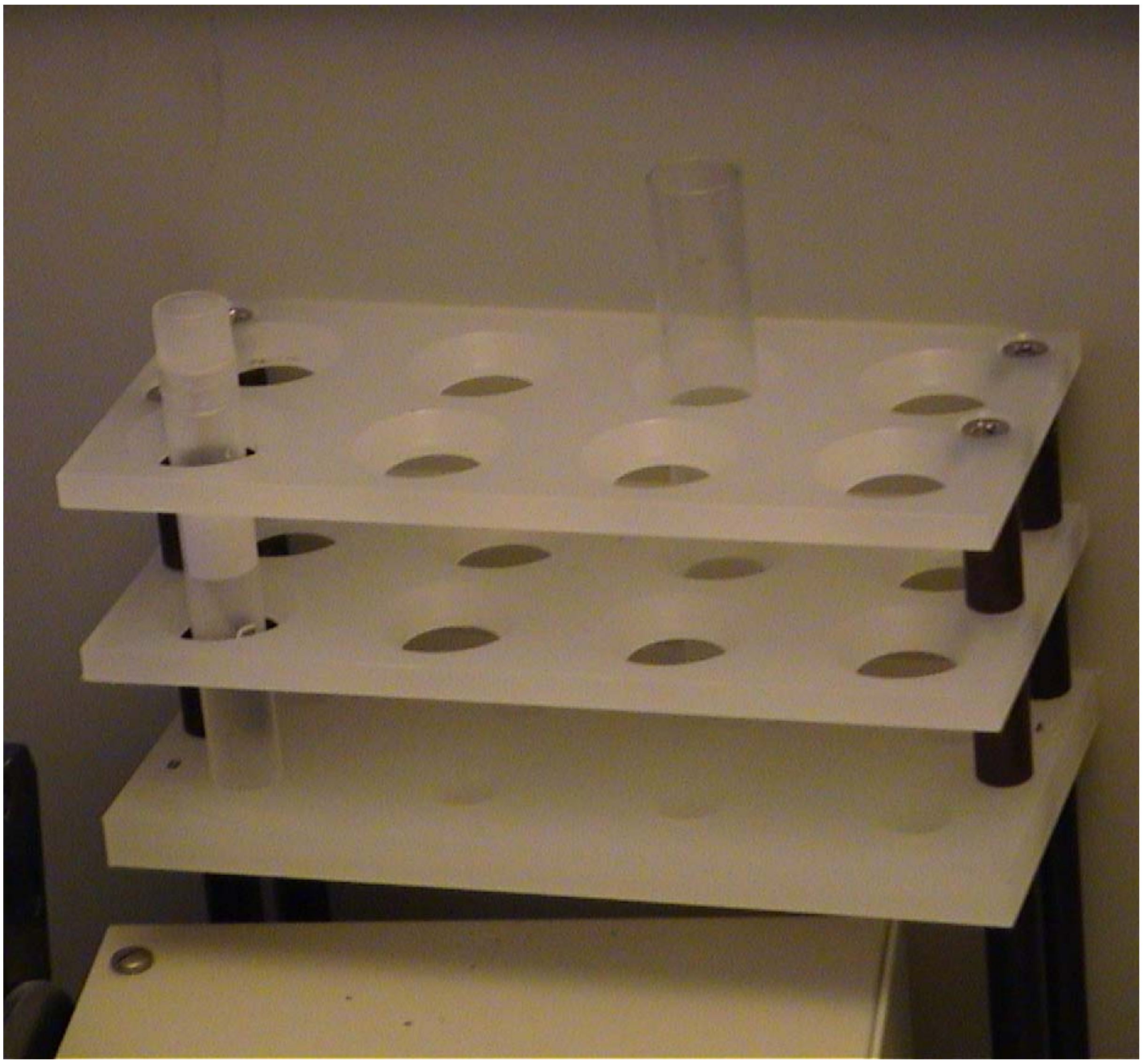
Variables:	RACK.2.INDEX = 1 to 50
	INITIAL.VOLUME.RACK.2 = 0 to 8 (ml)
	CONTAINER.CAPPED.RACK.2 = YES or NO

Testing Program:	JOOCHAN.RACK.2

- display off
- get.hand.a
- prompt Inster source position (Rack 2): 1 to 50
- input rack.2.index
- get.from.rack.2
- prompt Insert destination position (Rack 2): 1 to 50
- input rack.2.index
- put.into.rack.2
- park.hand

2 - Description of each PySection

✐ **2.06- Purpose:** Rack 16 x 100 (Rack F).



2 - Description of each PySection

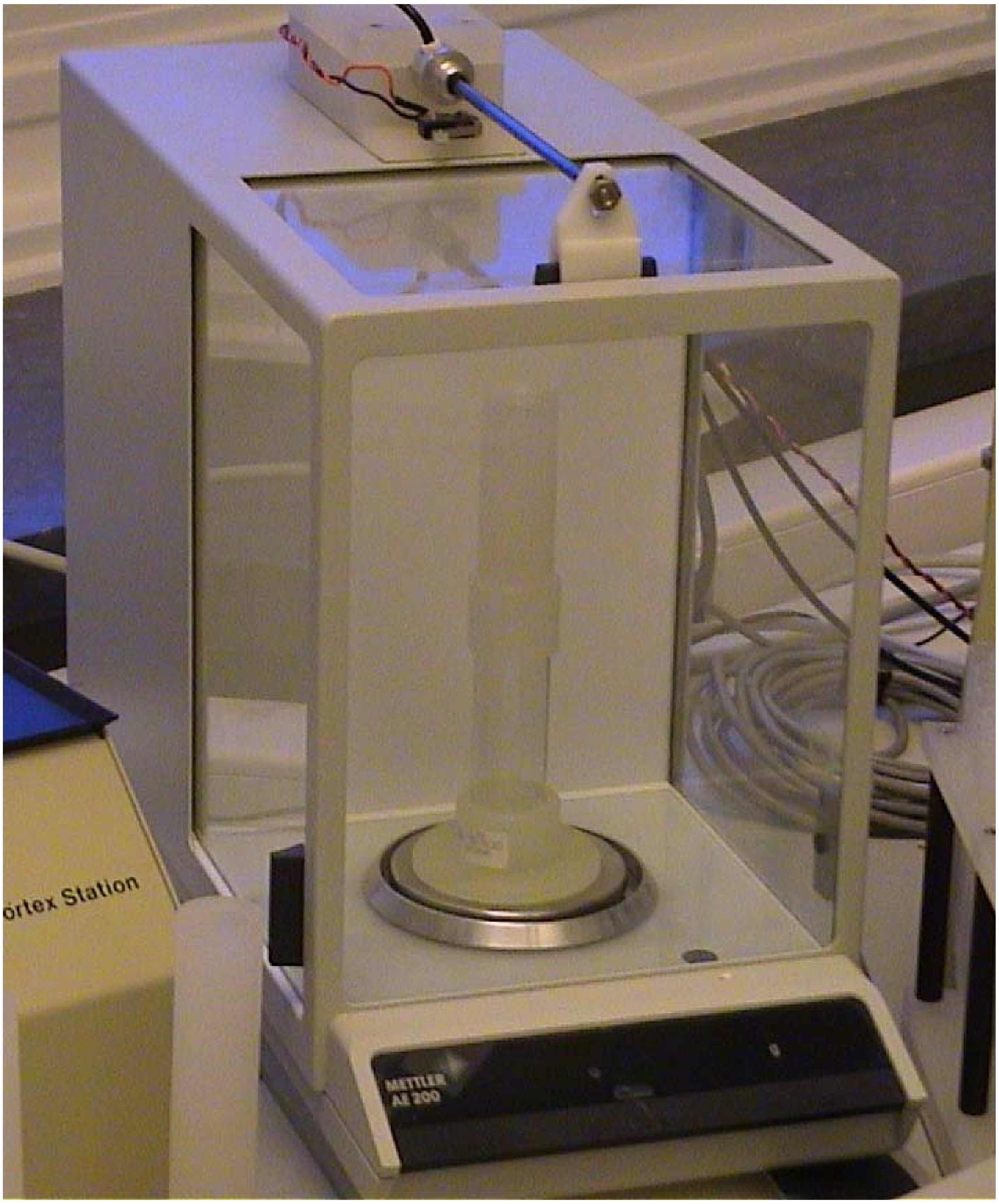
Description:

This Rack PySection provides storage for 16 x 100 mm test tubes. Each container in the rack has an associated volume and capped status that is automatically updated as the container is moved around the benchtop. The rack is made of polypropylene and is mounted on a snap-base for easy removal and replacement. Containers are held uprightin the rack with adequate spacing to permit individual container access with Zymate Hands.

Available commands:	F:GET.TUBE.FROM.RACK F:PUT.TUBE.IN.RACK GET.FROM.CENTRIFUGE PUT.INTO.CENTRIFUGE
Variables:	F:RACK.INDEX = 1 to 8
Testing Program:	(none)

2 - Description of each PySection

✐ **2.07- Purpose:** Weighing - Liquid Transfer – 16 x 100 mm Test Tubes.



2 - Description of each PySection

Description:

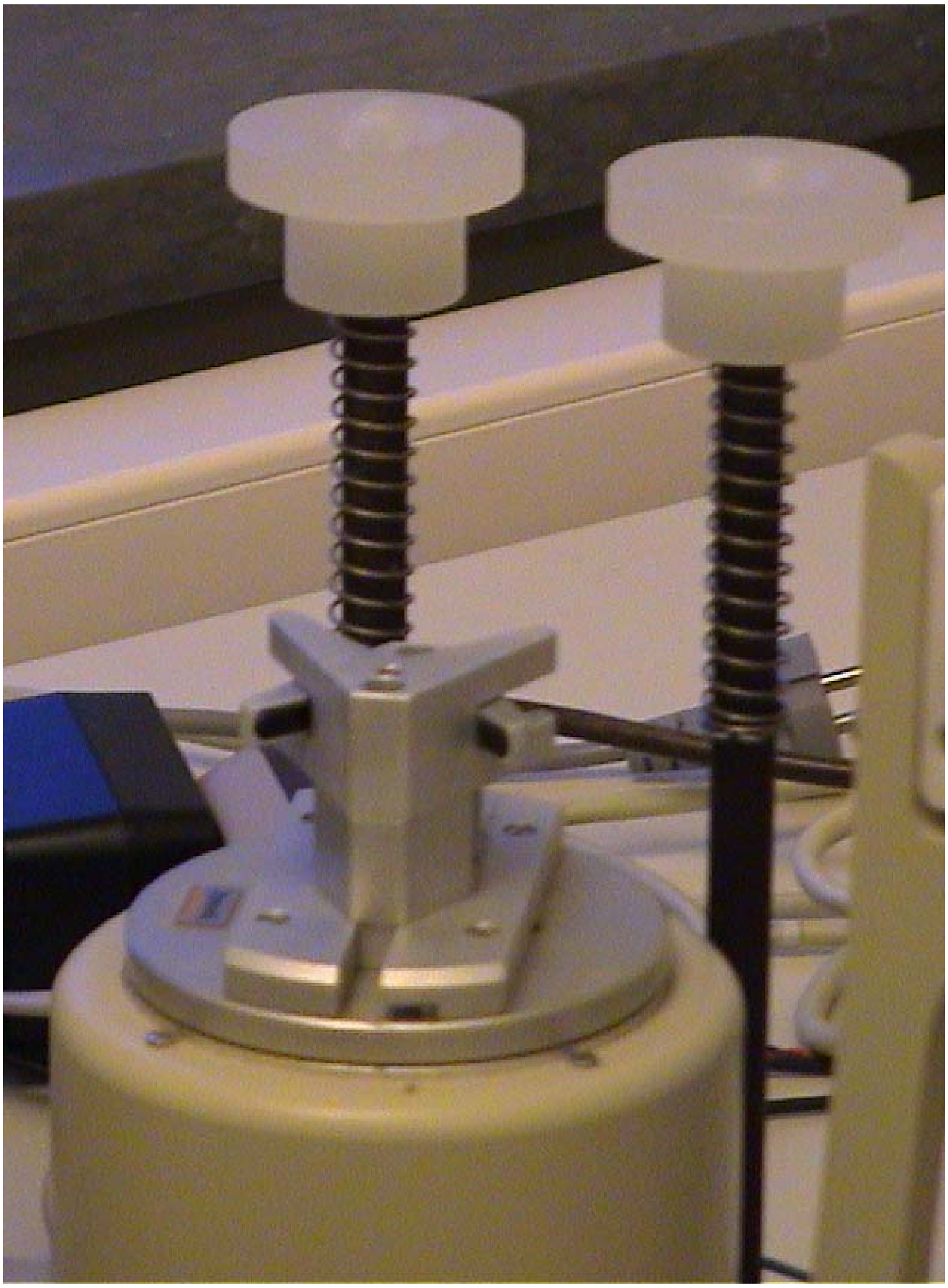
This PySection is used for analytical weighting of liquid samples and sample containers. The balance required by this PySection is a Mettler AE200 Series Analytical balance. A pneumatic Balance Door Opener included in this PySection allows sample containers to be top-loaded into the balance, preventing drafts from affecting the balance. Thansfer of liquid samples is done by pippetting.

Available commands:	PUT.INTO.BALANCE OBTAIN.WEIGHT GET.FROM.BALANCE MOVE.OVER.BALANCE BD:OPEN.DOOR BD:CLOSE.DOOR BD:TARE
Variables:	WEIGHT.VALUE (gr) TARGET.WEIGHT
Testing Program:	JOOCHAN.BALANCE

- display off
- get.hand.a
- prompt Inster source position (Rack 1): 1 to 50
- input rack.1.index
- get.from.rack.1
- put.into.balance
- obtain.weight
- prompt Weight of the tube:
- printc weight.value
- get.from.balance
- prompt Insert destination position (Rack 1): 1 to 50
- input rack.1.index
- put.into.rack.1
- park.hand

2 - Description of each PySection

✐ **2.08- Purpose:** Capping – 16 x 100 mm tubes.



2 - Description of each PySection

Works well with 11 x 100 mm test tubes.

Description:

This Capping PySection provides automated capping and uncapping of screw-capped 16 x 100 mm test tubes. The success of capping and uncapping operations is internally verified. Preset capping conditions cap containers at ½ torque. Default conditions also cause caps not to be discarded after uncapping operations. In the event that two containers are in an uncapped state at the same time, such as in liquid/liquid extraction operations, two cap holders are provided. The Capping PySection has been preset to use cap holder 1. If required, the station can act as a container holder. Also, an uncapped container positioned in the station can act as a sample source or destination for pippetting operations.

Available commands:	PUT.INTO.CAPPER CAP UNCAP GET.FROM.CAPPER MOVE.OVER.CAPPER

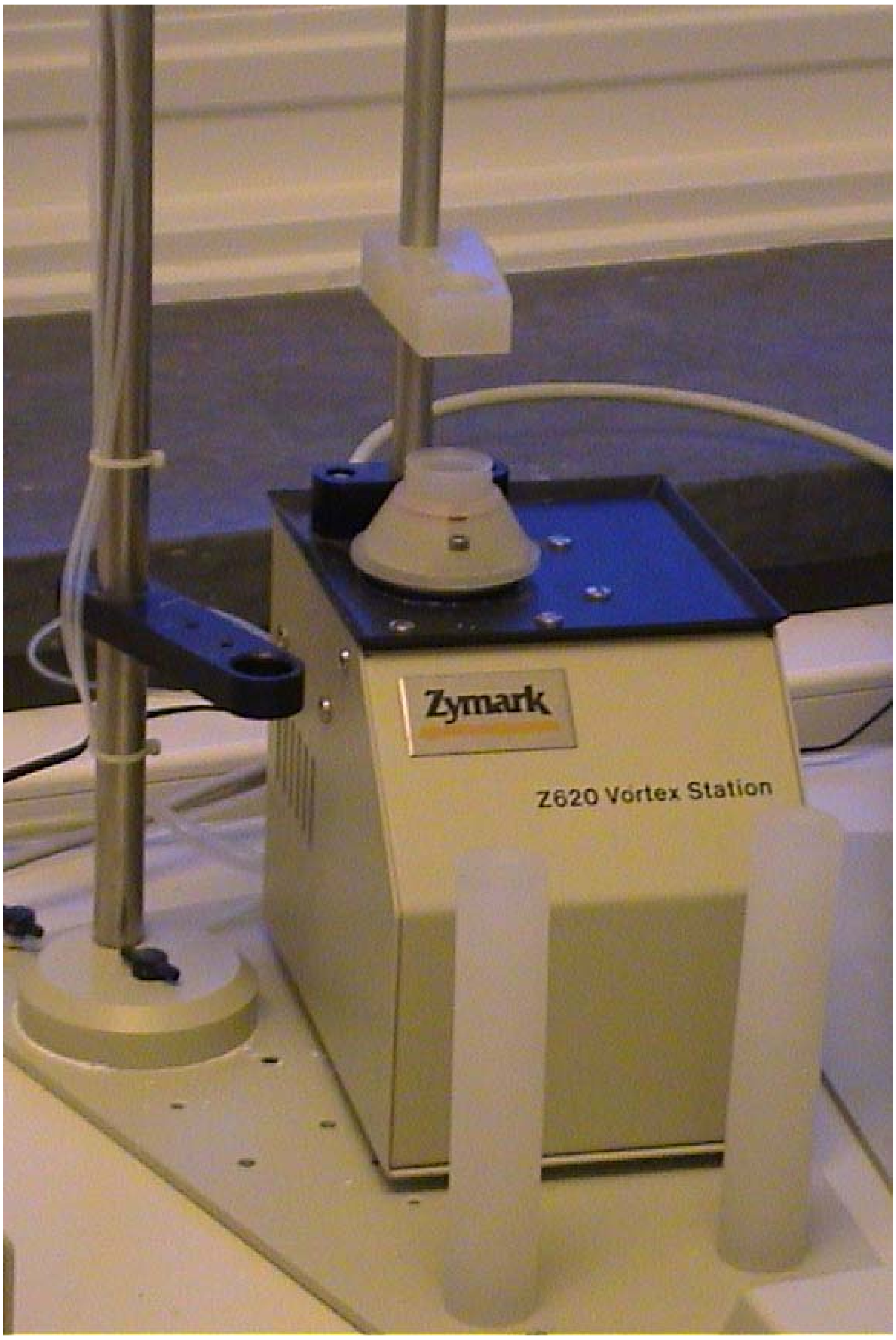
Variables:	CAP.INDEX = 1 or 2
	DISCARD.CAP = YES or NO

Testing Program:	JOOCHAN.CAPPER

- display off
- get.hand.a
- get.from.centrifuge
- put.into.capper
- uncap
- get.from.capper
- put.into.capper
- cap
- get.from.capper
- prompt Inster destination position (Rack 2): 1 to 50
- input rack.2.index
- put.into.rack.2
- park.hand

2 - Description of each PySection

✐ **2.09- Purpose:** Dilute and Dissolve – 16 x 100 mm Tubes (Vortex).



2 - Description of each PySection

Description:

The Dilute & Dissolve PySection is used to quantitatively dispense up to 3 discrete liquids into a container. Liquid addition can be followed by a vigorous vortexing action to speed mixing and dissolving. Liquid addition without vortexing and vortexing without liquid addition is also possible. This PySection may be recognized as the first or only Dilute & Dissolve PySection, or, as the second Dilute & Dissolve PySection within the bench configuration.

Available commands:	PUT.INTO.VORTEX VORTEX.ON VORTEX.OFF VORTEX.TIMED.RUN GET.FROM.VORTEX MOVE.OVER.VORTEX DILUTE

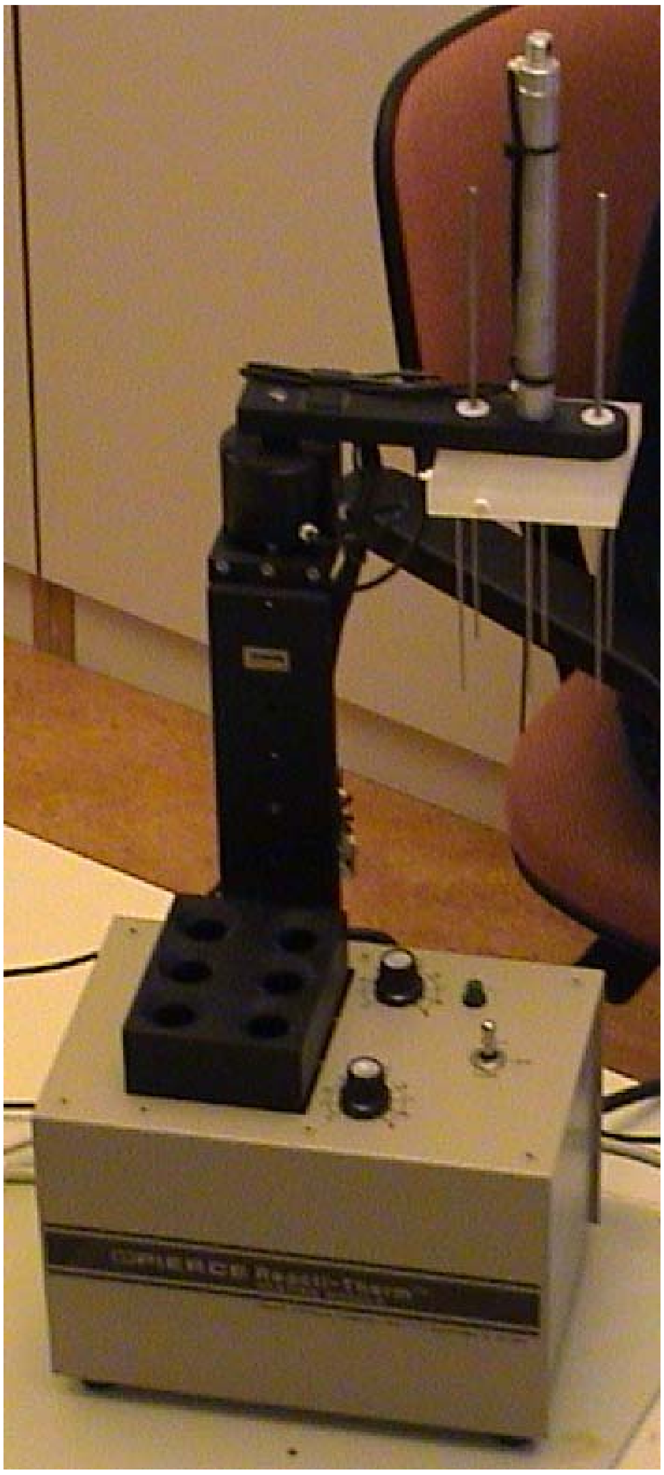
Variables:	VORTEX.TIME = 0 to 5240 (sec)
	VORTEX.SPEED.1 = 0 to 200

Testing Program:	JOOCHAN.VORTEX

- display off
- get.hand.a
- prompt Insert source position (Rack 1): 1 to 50
- input rack.1index
- put.into.vortex
- prompt Insert duration time:
- input vortex.time
- vortex.timed.run
- get.from.vortex
- prompt Insert destination position (Rack 1): 1 to 50
- input rack.1.index
- put.into.rack.1
- park.hand

2 - Description of each PySection

✐ **2.10- Purpose:** Evaporation.



2 - Description of each PySection

Description:

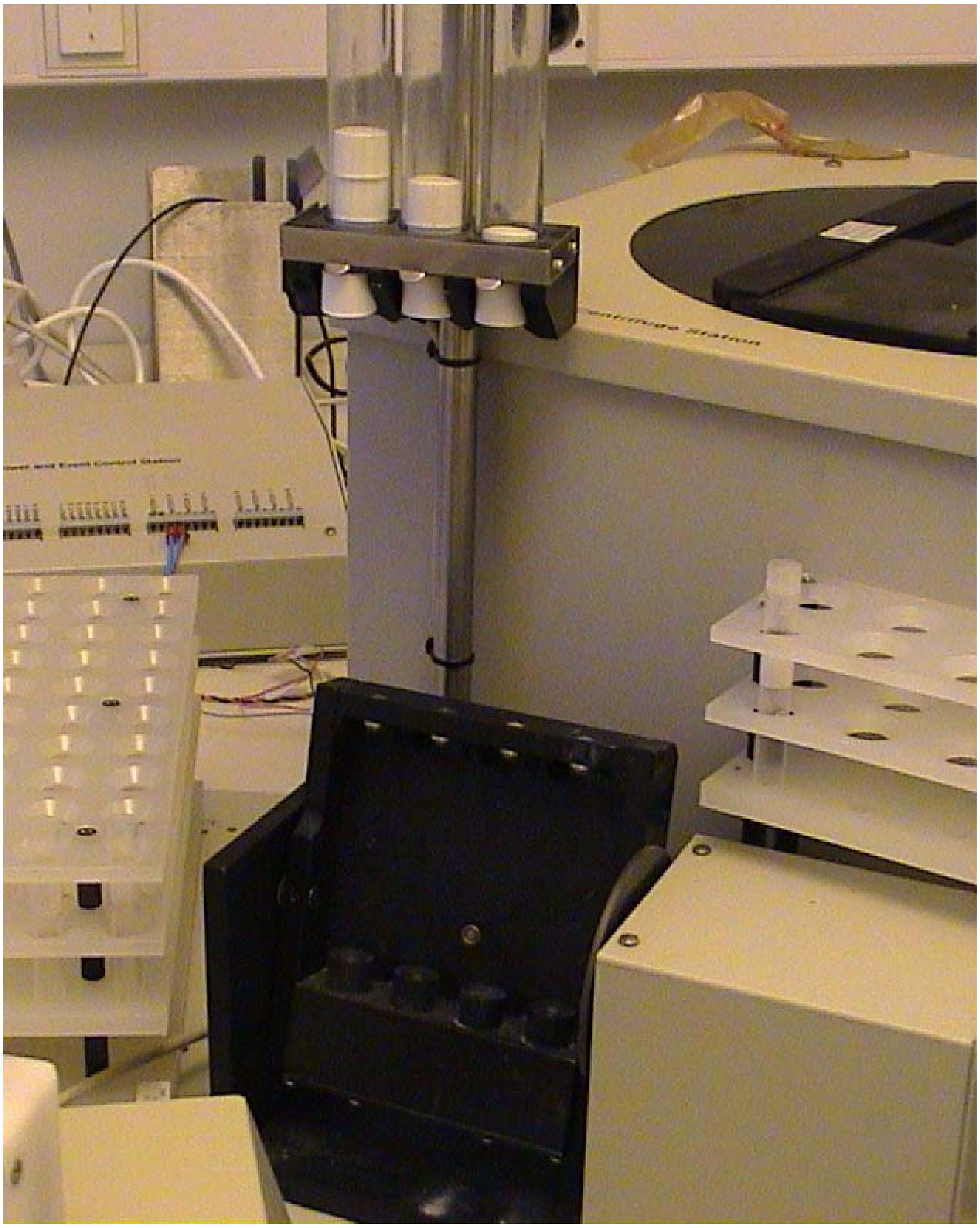
The Evaporator PySection for 16 x 100 mm test tubes provides for the evaporation of solvent from containers placed within its heating block. The temperature-controlled heating block is settable between ambient and 100 ºC. Temperature is manually set by user. Solvent evaporation is accelerated by gas purge tubes which are lowered into the containers placed in the heating block. The user-supplied purge gas is used to pneumaticaly operate the station as well as dry the samples. Gas is conserved by being turned on only while there are samples in the station. Evaporation to dryness is not detectable by this station. Experimentation should be performed to determine the evaporation time necessary of the sample. This time can vary with the sample's position in the block, how many samples are in the block, block temperature, gas flow rate, solvent type, solvent volume container, container shape, etc.
The heating block is normaly plugged into its own power source and remains on at its set temperature while its power is on. If it desired to turn the heating block on and off under Zymate System Control, the station may be plugged into one of the Power & Event Controller's AC outlet, then commands controlling that outlet and timer statements to allow sufficient block warm-up may be added to the top-level program.

Available commands:	PUT.INTO.EVAPORATOR GET.FROM.EVAPORATOR
Variables:	EVAPORATOR.INDEX = 1 to 6
Testing Program:	JOOCHAN.EVAPORATOR

- display off
- get.hand.a
- prompt Insert source position (Rack 1): 1 to 50
- input rack.1.index
- get.from.rack.1
- prompt Insert evaporator position: 1 to 6
- input evaporator.index
- put.into.evaporator
- pause5
- get.from.evaporator
- prompt Insert destination position (Rack 1): 1 to 50
- input rack.1.index
- put.into.rack.1
- park.hand

2 - Description of each PySection

✐ **2.11- Purpose:** Tumble Mixer – 16 x 100 mm Tube.



2 - Description of each PySection

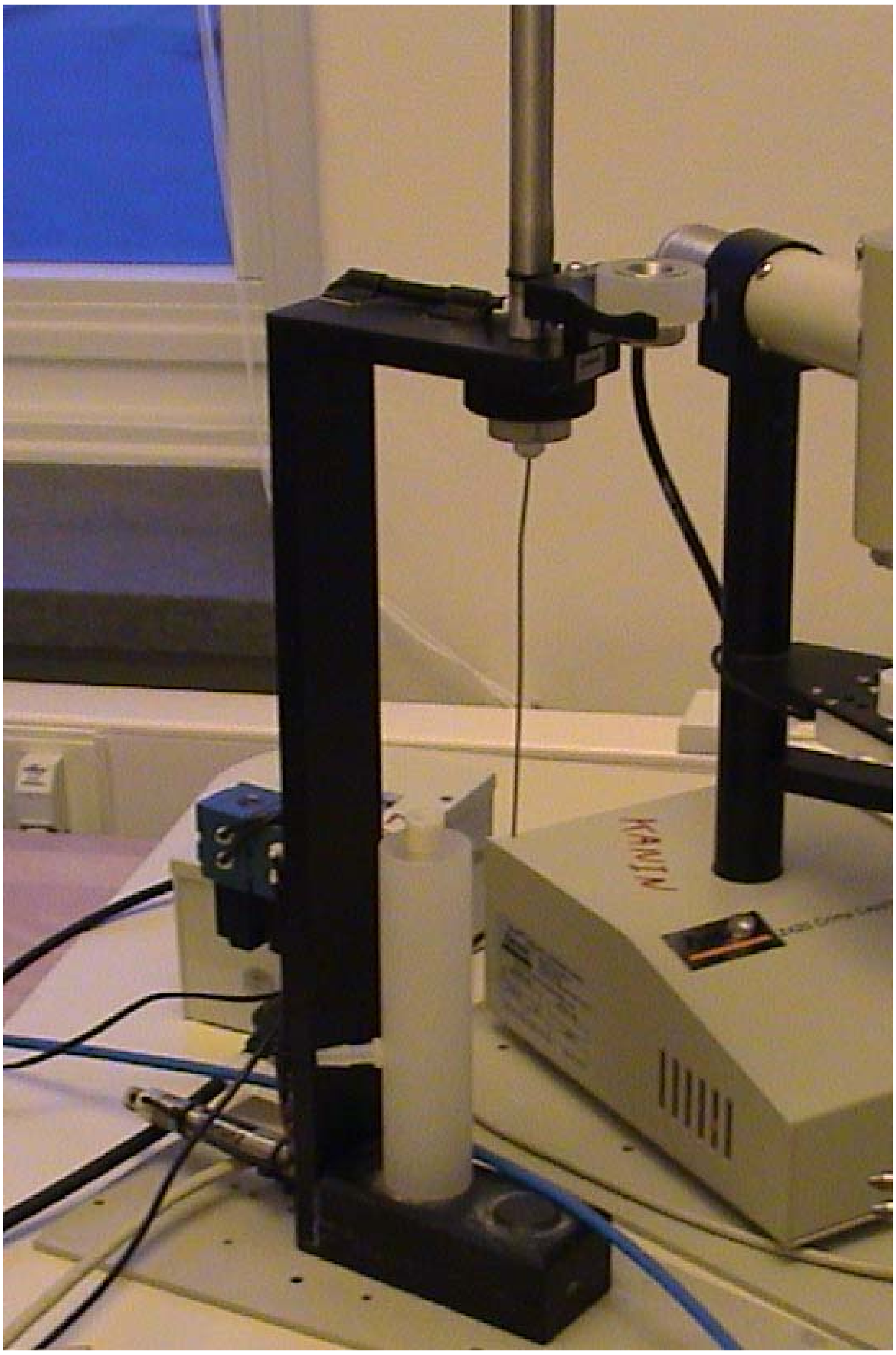
Description:

This PySection provides gentle but thorough mixing of samples in 16 x 100 mm test tubes. The tumble mixer is typically used for liquid/liquid extraction procedures. The tumbling action optimizes the rate of sample partitioning between the solvent phases, while minimizing emulsion formation. Sample tubes with slip-on caps, held in place by the tumbling station via a spring-loaded mechanism, are turned end over end at a controlled 15 rpm. Verification techniques ensure that the sample tubes are capped before they are placed into the tumbling station. Included in this PySection is the four-position tumbling station, a slip-on cap dispenser with a capacity of 150 caps and a cap removal station. The optional addition of a centrifuge allows quick, effective phase separation and breaking of residual emulsions. Th ZP710 Centrifuge fits behind the Tumble Mixing PySection so that no additional bench space is required.

Available commands:	PUT.INTO.TUMBLE.MIXER GET.FROM.TUMBLE.MIXER
Variables:	MIXER.INDEX = 1 to 4
Testing Program:	(none)

2 - Description of each PySection

✐ **2.12- Purpose:** Liquid/Liquid Extraction.



2 - Description of each PySection

Description:

This PySection performs the transfer of the desired extraction layer from an extraction container to a fluid-filled holding loop for subsequent dispensing into a collection container or waste. The extraction and collection containers are held by the Robot. You determine and program the depth at which the cannula goes into the extraction container, the volume to be extracted and the time required to extract the sample. The pneumatically controlled cannula wash shuttle, rinses the cannula internally and externally with a solvent volume specified in the application program.

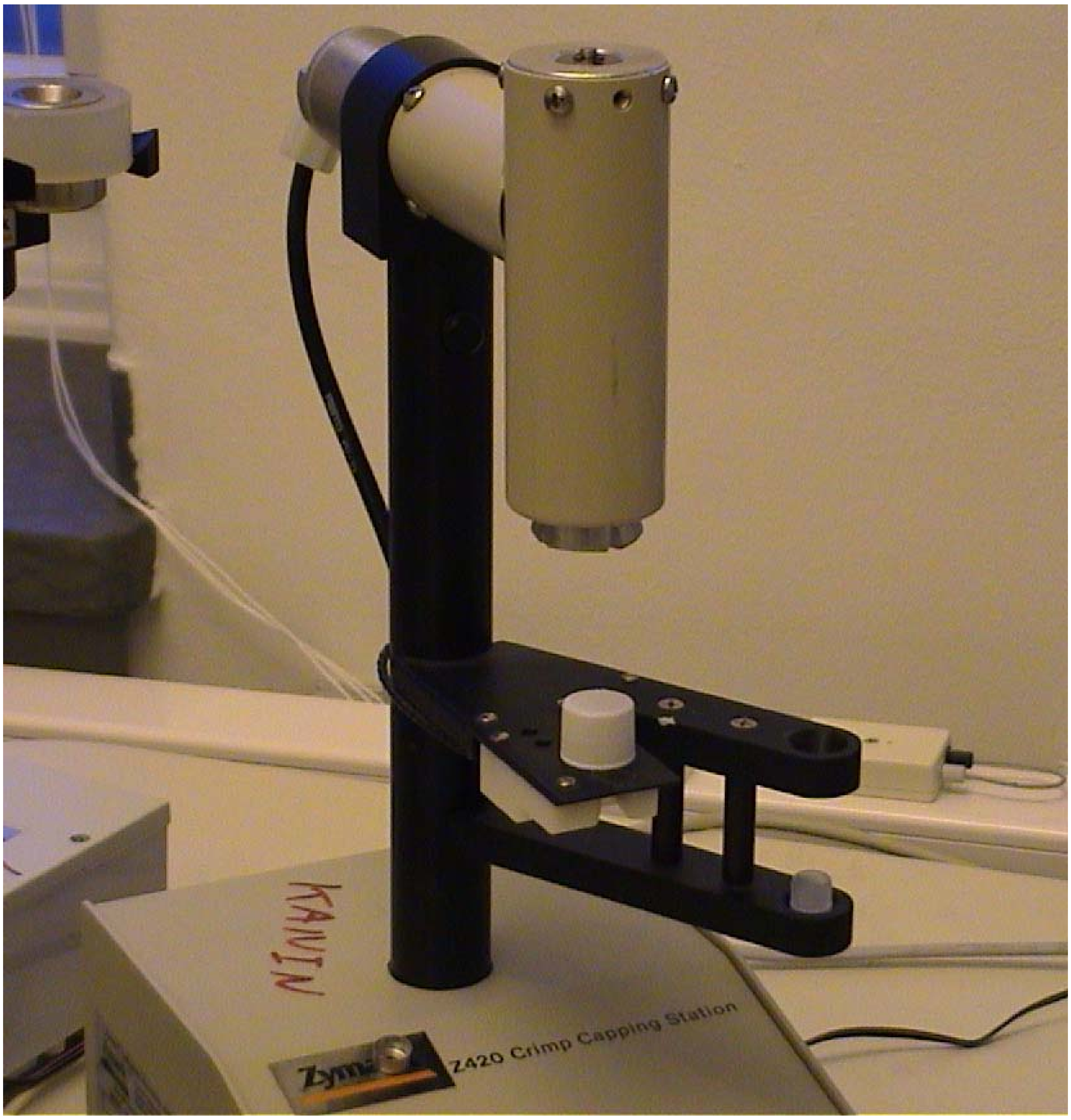
Available commands:	ASPIRATE.LL.CANNULA DISPENSE.LL.CANNULA WASH.LL.CANNULA

Variables:	LL.CANNULA.DEPTH = 0 to 15 (cm)
	LL.CANNULA.VOLUME = 0 to 9.8 (ml)
	LL.ASPIRATE.TIME = 1 to X (sec)
	LL.WASH.VOLUME = 1 to X (ml)

Testing Program:	(none)

2 - Description of each PySection

✐ **2.13- Purpose:** Crimp capping.



2 - Description of each PySection

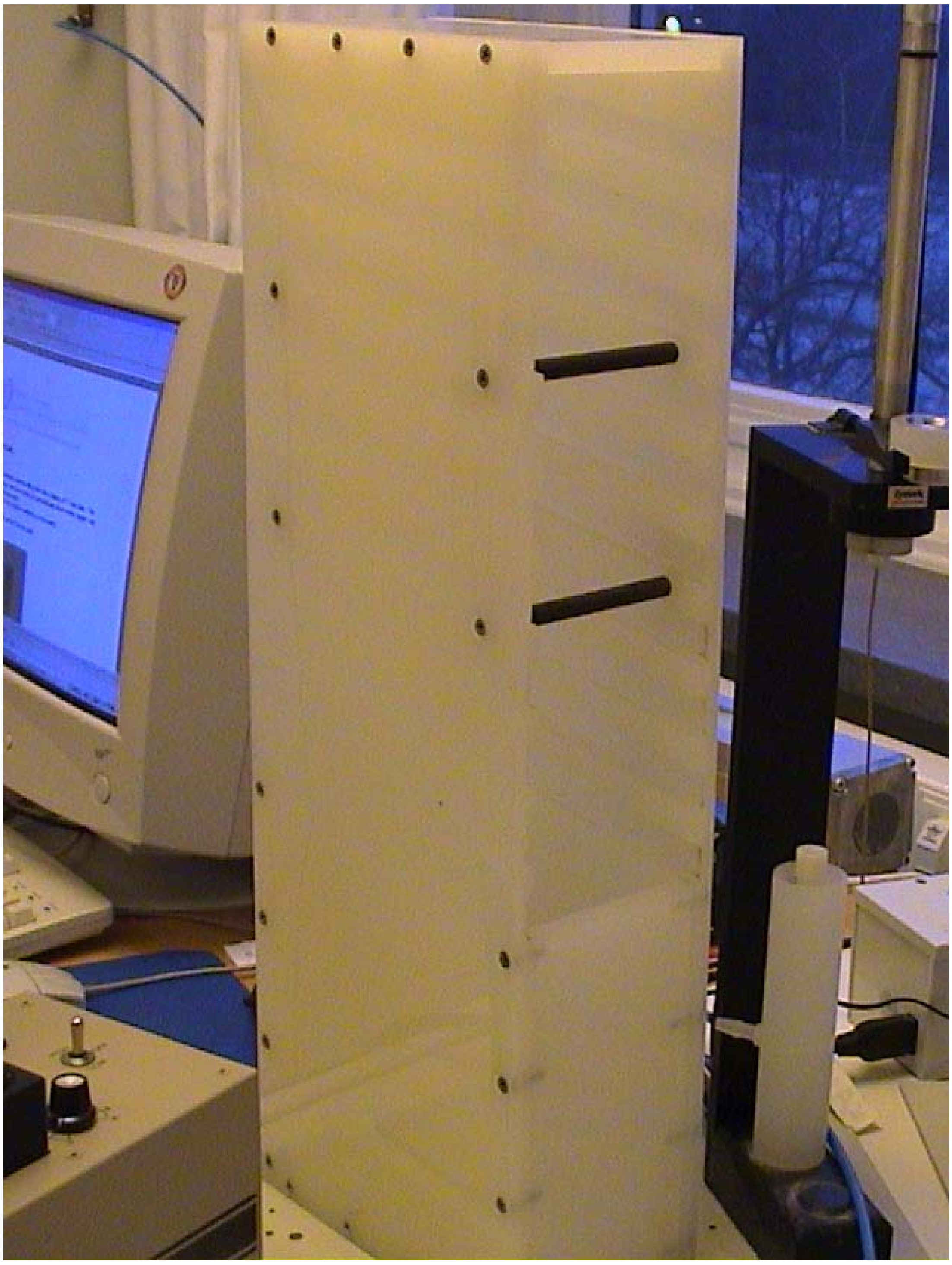
Description:

This PySection provides automated filling and capping of 11 mm vials. These vials are generally used with GC or HPLC autosamplers. The Crimp Capping PySection consists of a pneumatically-driven Crimp Capper, vial cap holder, vial holder and tip guide for vial filling operations. Cap presence in the holder is verified by a microswitch.

Available commands:	PUT.INTO.CRIMP.CAPPER CRIMP.VIAL UNCAP.VIAL GET.FROM.CRIMP.CAPPER MOVE.OVER.CRIMP.CAPPER
Variables:	(none)
Testing Program:	(none)

2 - Description of each PySection

✐ **2.14- Purpose:** Test Tube Dispenser – 16 x 100 mm test tubes.



2 - Description of each PySection

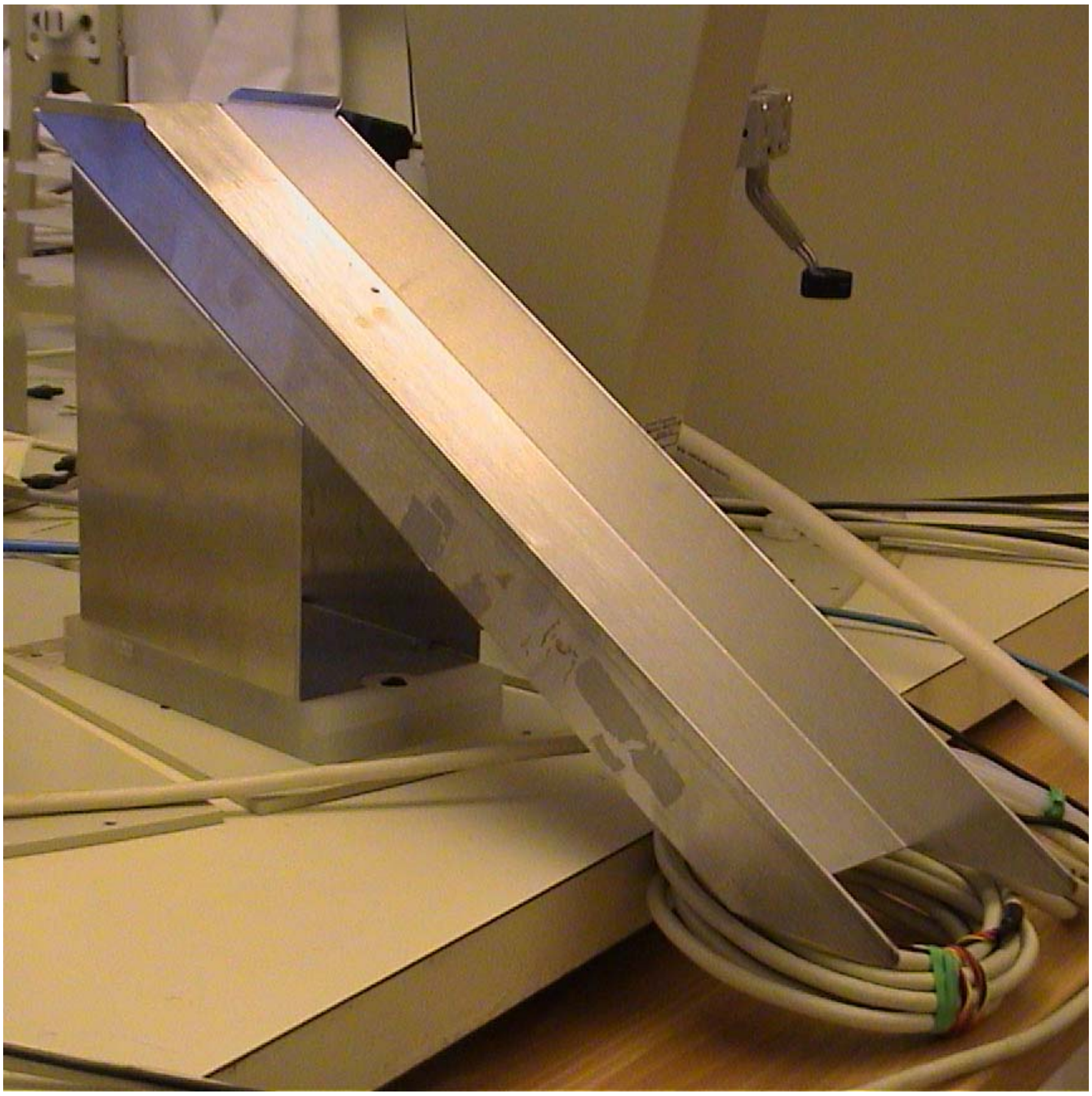
Description:

The Tube Dispenser PySetcion is used to dispense 16 x 100 mm test tubes. The dispenser housing has slots for up to ten stainless steel trays each having a capacity for 35 tubes. Included with this PySection is the dispenser housing and two trays. Additional trays may be purchased in set of two.

Available commands:	GET.FROM.DISPENSER.1
Variables:	(none)
Testing Program:	(none)

2 - Description of each PySection

✐ **2.15- Purpose:** Disposal.



2 - Description of each PySection

Description:

This PySection provides for the discarding of used disposables such as pipet tips, extraction columns, screw caps, and disposable filters. The PySection consists of a chute that may be positioned to overhang the benchtop. A waste receptacle can be placed under this overhang or a bag can be attached to receive the discarded items.

Available commands:	DISPOSE.TO.WASTE

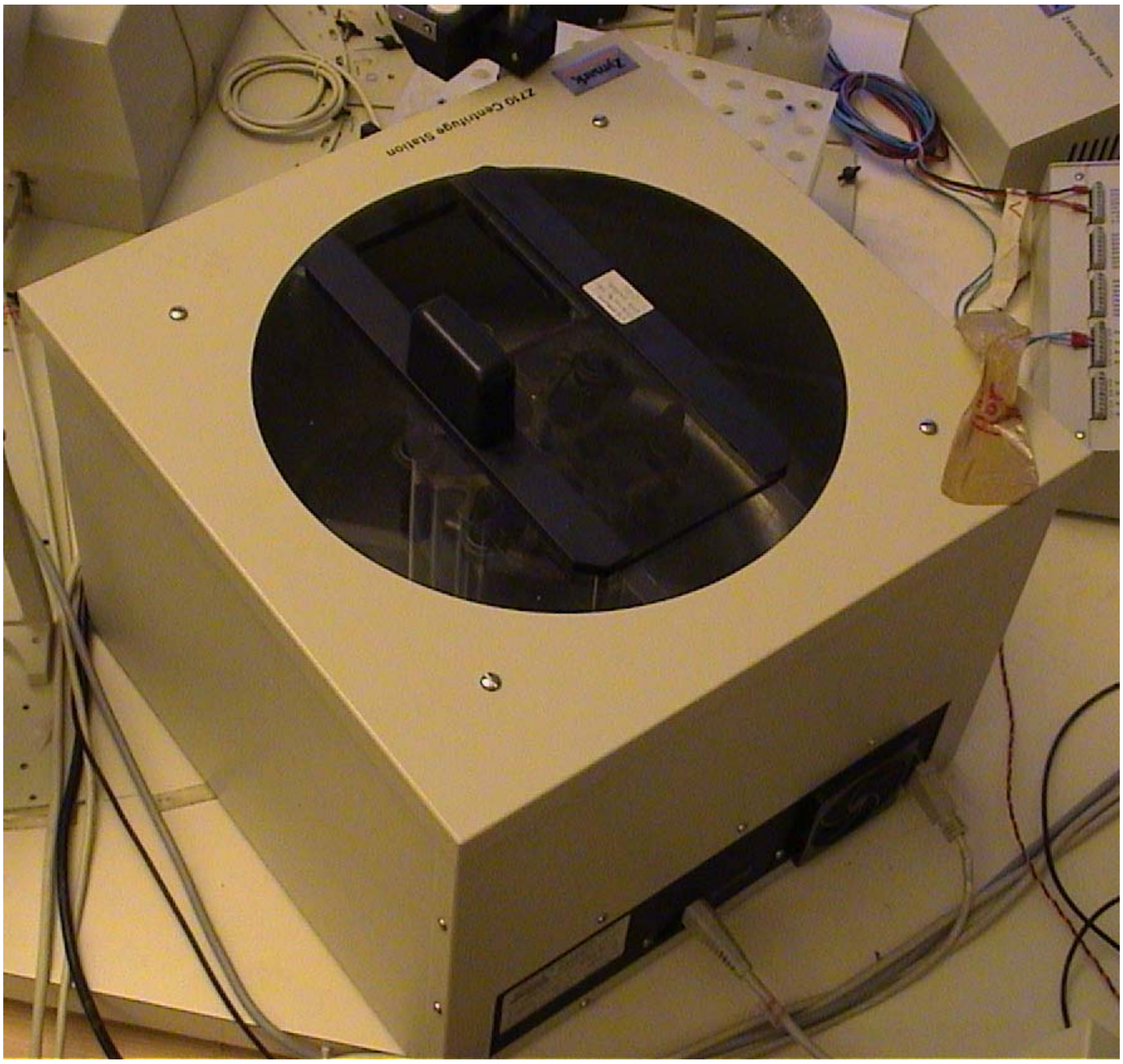
Variables:	GRIPPER.HAND.WASTE.POS = 0 to 200
	SYRINGE.HAND.WASTE.POS = 0 to 200

Testing Program:	JOOCHAN.DISPOSAL

- display off
- get.hand.a
- prompt Inster source position (Rack 1): 1 to 50
- input rack.1.index
- get.from.rack.1
- dispose.to.waste
- park.hand

2 - Description of each PySection

✐ **2.16- Purpose:** Centrifuge - 16 x 100 mm Tubes.



2 - Description of each PySection

Description:

This PySection provides automated centrifugation of samples. The electro-mechanically-operated loading door and rotor position indexing capability of the centrifuge allow robotic interaction with this station. The 8-position "balancing container rack" provides two transfer positions for placing samples into and removing samples from the centrifuge, and six position for holding any balance container not in use during centrifugation. The sample's position within the centrifuge and its spin time is automatically tracked when the aplication is serialized using EasyLab Program Scheduler.

Available commands:	PUT.INTO.CENTRIFUGE GET.FROM.CENTRIFUGE

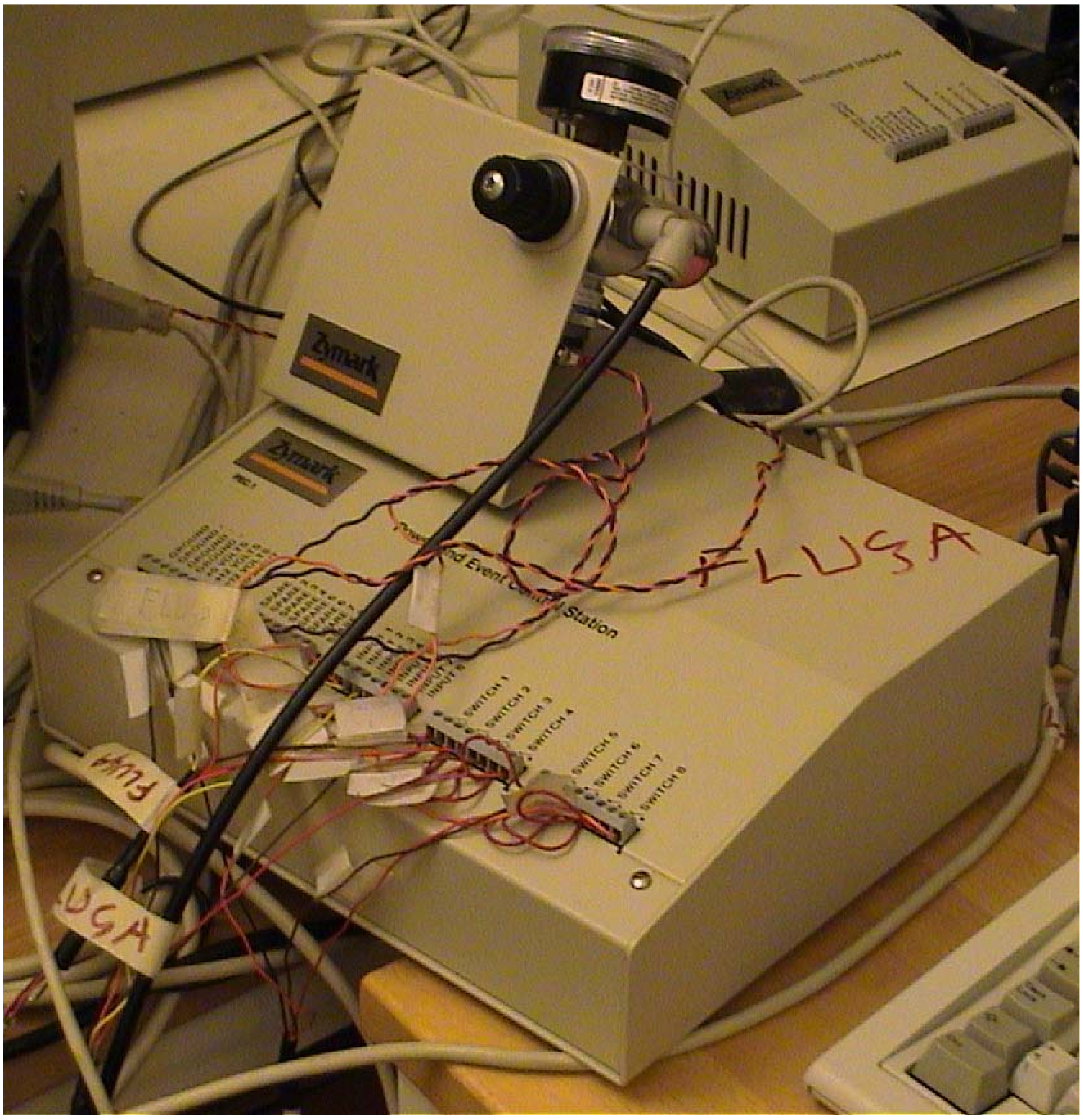
Variables:	CENTRIFUGE.INDEX = 1 to 6
	CENTRIFUGE.SPEED = 0 to 3000

Testing Program:	JOOCHAN.CENTRIFUGE

- display off
- get.hand.a
- prompt Inster source position (Rack 1): 1 to 50
- input rack.1.index
- get.from.rack.1
- prompt Inster centrifuge position: 1 to 6
- input centrifuge.index
- put.into.centrifuge
- pause10
- f:vol.output=-1
- get.from.centrifuge
- prompt Inster destination position (Rack 1): 1 to 50
- input rack.1.index
- put.into.rack.1
- park.hand

2 - Description of each PySection

✐ **2.17- Purpose:** Power and Event Controller (PEC).



2 - Description of each PySection

Description:

The Power and Event Controller (PEC) interfaces with devices that supplement the operation of other PySections in the Zymate System. The power & Event Controller provides Zymate System control for devices used in Crimp Capping, Evaporation, Liquid/Solid Extraction, etc. The software supports the installation and operation of up to 5 Power & Event Controllers in the system. Each Power & Event Controller provides programmable control of 8 output switches, 8 logical inputs, 2 AC On/Off power outlets, 1 variable AC power outlet and an analogic/digital input. A +5V, +12V and –12V power supply is also included. There are 8 Spare connectors also available for use. These are used as terminals when an internal option such as the Preamplifier Module is installed within the PEC. The Spare connectors can also be used as tie points if no option is installed. A Power & Event Controller's facilities may be utilized by different PySections. The assignment of a Power & Event Controller and its programmable connections takes place when the PySection requiring their usage is installed in the system. The Power & Event Controller facilities are operated by the individual PySections.
An Air Monitor Manifold that provides a confirm sensor, filter, regulator and air line connections manifold is included with the Power and Event Controller. Only one Air Monitor Manifold is required, regardless of how many ZP830's are to be used.

Available commands:	PEC.n.SWITCH.m.ON (n = 1 to 5, m = 1 to 8) PEC.n.SWITCH.m.OFF (n = 1 to 5, m = 1 to 8) PEC.n.ACm.ON (n = 1 to 5, m = 1 to 2) PEC.n.ACm.OFF (n = 1 to 5, m = 1 to 2) PEC.n.A.TO.D.UNSCALED (n = 1 to 5)

Variables:	PEC.n.VAR.AC = x (n = 1 to 5, x = 0 to 200)
	PEC.n.INPUT.m (n = 1 to 5, m = 1 to 8)
	CONFIRM.AIR.SENSOR = YES or NO

Testing Program:	(none)

2 - Description of each PySection

✐ **2.18- Purpose:** Zymate II Core System.

2 - Description of each PySection

Description:

The Zymate II Core System provides the basic functions of a PyTechnology System. The Controller with its EasyLab language allows you to construct application programs using a series of PySections commands and variables setting statements. The Zymate II Robot Module provides highly efficient sample handling and transfer between PySections. The Zymate II Robot is pre-mounted on a central locator plate around which the PySections are arranged. The Zymate Printer provides a hardcopy of system conditions and data. The EasyLab Program Scheduler Module automates the processing of multiple samples in a serialed manner. The Remote Robot Teaching Module is used for positioning the Zymate Robor without having to be at the keyboard.

Available commands:	ROBOT.EXERCISE INITIALIZE.SYSTEM SHUTDOWN.SYSTEM PRINTER.ON PRINTER.OFF PARK.HAND

Variables:	ERROR.CHECK = YES or NO
	SHUTDOWN.ON.ERROR = YES or NO
	PRINT.CONDITIONS = YES or NO
	SPEED = 0.20 to 1.00

Testing Program:	(none)

3 – Advanced operations

✎ 3.1- Purpose: Re-teaching positions.

🕮 Description:

When a PyPlate is removed – allowing the station to be placed in a different location than when mounted before – the station positions must be retaught to the robot.

🗎 Procedures:

💣 Using and example for better understanding → Move HAND.A from position 47 to position 21.

1. Load the proper dictionary to the controller.
2. Put the PySection Disk labelled “GP HAND A” in the controller.
3. Go to System → PyAppend dictionary.
4. Chose the proper dictionary (in our case named CP900-1.ZYD).
5. Input the destination sector location number (we have chosen sector 21).

💣 You can find the sector number by sighting down the right-hand edge of the station (while facing the Robot) and determining the nearest sector number on the Robot locator plate).

6. After a while, press Y to run the SETUP program.
7. When asked, press Enter if the station is mounted on the PyPlate.

💣 You can install a PySection without having it physically installed.

8. Press Enter to TEST the station.
9. If everything goes well, the station will work properly in the new position.

4 – Loop, conditional & pausing commands

4.1- Purpose: DO / ENDDO.

Description:

"DO Loops" are used within a program to execute a group of statements a specific number of times.

Format:

```
DO x TIMES         x = number of executions to perform
      .   ─┐
      .    ├──▶   program
      .    │
ENDDO     ─┘
```

Remarks:

- The DO and ENDDO statements "frame" the section of program they affect.
- All DO statements is not affected by an ENDDO statement within the same program. This is, they are program-level sensitive.
- An active DO statement is not affected by an ENDDO statement appearing in another program.
- If a number is calculated to specify the number of loops to execute, and that number is a non-integer value, that value is rounded to the closest integer.
- Under DISPLAY OFF, the number of the loop being executed is still displayed.
- DO Loops can be exited (using a conditional or GOTO statement) before the loop has completed.

Example Program: JOOCHAN.DEMO.LOOP

- ➢ display off
- ➢ rack.1.index=1
- ➢ evaporator.index=1
- ➢ get.hand.a
- ➢ do 6 times
- ➢ get.from.rack.1
- ➢ put.into.evaporator

4 – Loop, conditional & pausing commands

- rack.1.index=rack.1.index+1
- evaporator.index=evaporator.index+1
- enddo
- rack.1.index=rack.1.index-1
- evaporator.index=evaporator.index-1
- do 6 times
- get.from.evaporator
- put.into.rack.1
- rack.1.index=rack.1.index-1
- evaporator.index=evaporator.index-1
- enddo
- park.hand
- rack.1.index=rack.1.index+1
- evaporator.index=evaporator.index+1

4 – Loop, conditional & pausing commands

4.2- Purpose: GOTO.

Description:

This command is used in a EasyLab statement to branch unconditionally out of the normal program sequence to a specified line number.
GOTO statements are used primarily to alter the sequence of operation. This involves jumping to another location in the program to execute another program or module action, or, pausing system operation.

Format:

GOTO x	x = constant or variable representing a line number

Remarks:

- This command affects the sequence of execution of EasyLab program statements. A GOTO statement causes program execution to "jump" to the statement in that the program that is labelled with the specified line number. Program execution continues from that statement.
- EasyLab Program lines are executed in the ordeer they appear in the program. Although line numbers are not required to determine the operational flow of the program, they are used for jumping purposes. These numbers which can range from 1 to 65535 are used only as reference points for program branching.
- GOTO statements are program-level sensitive. A GOTO statement must refer to a numbered statement whitin the same program.
- The numbered statement referred to by a GOTO statement is always searched for from the beginning of that program.
- GOTO statements should be programmed logically to avoid creating endless loops.
- If the statement number is calculated using an EasyLab math expression and a non-integer value results, that calculated value is rounded to the nearest integer. The integer value must range between 1 and 65535.
- A line number specified by a GOTO statement mus be labeled a statement appearing within *that* program.
- The same line number may be used in several different programs. Branching always occurs within the same program.

4 – Loop, conditional & pausing commands

- If the same line number appears multiple times in a single EasyLab program, the first occurrence of that number from the beginning of the program is the one referenced.
- There are 5 spaces reserved at the beginning of each statement for line numbers.
- A line number may appear before each statement in EasyLab program, but a program may be written without any line numbers.
- A line number does NOT cause an EasyLab program to be excluded by sequential line numbers.
- Line numbers are used as reference markers for program braching statements.

Example Program: JOOCHAN.DEMO.GOTO

```
➢  1   display off
➢      prompt Write "YES" to see the time or "NO" to exit
➢      input joochan.time
➢  10  if joochan.time=0 then 100
➢      ? clock
➢      goto 1
➢  100
```

4 – Loop, conditional & pausing commands

4.3- Purpose: IF / THEN.

Description:

This command is used in an EasyLab statement to make a decision regarding program flow based on a result returned from an expression. The change in program flow can incolve jumping to another program line, executing another program or module action, performing a calculation, specifying a module action, or pausing system operation.
Normally, EasyLab program lines are executed in the order they appear in the program. When the IF/THEN statement involves jumping ro another program line, a *line number* is used. Line numbers (which can range 1 to 65535) are used only as reference points for program branching by IF/THEN and GOTO statements.

Format:

IF *expression comparator expression* THEN *statement*

where *expression* is either a constant, a real data variable, or a module command variable (input or bi-directional only)

where *comparator* is either =, <>, >, <, <= or >=

where *statement* is either a line number, module command, program, math expression, or an EasyLab language command (except an IF/THEN statement or DO LOOP)

Remarks:

- If the condition is not satisfied, the statement following “THEN” is ignored. Program execution continues with the next statement in the EasyLab program.
- IF/THEN statements are program-level sensitive when the statement following “THEN” refers to a numbered statement. When “THEN” is followed by a number, the numbered statement must appear within the same program as the IF/THEN statement and be preceded with the line number specified.
- If a line number is calculated following “THEN”, and a non-integer value results, that value is rounded to the nearest integer.

4 – Loop, conditional & pausing commands

- If a line number is calculated following "THEN", the "THEN" must be followed by "GOTO".
- If a line number is specified following "THEN", "GOTO" is not necessary if a constant is used.
- A line number specified by an IF/THEN statement must label a statement apearing within THAT program.
- The same line number may be used in several different programs. Branching always occurs within the same program.
- If the same line number appears multiple times in a single EasyLab program, the first occurrence of that number from the beginning of the program is the one referenced.
- There are 5 spaces reserved at the beginning of each statement for line numbers.
- A line number may appear before each statement in EasyLab program, but a program may be written without any line numbers.
- A line number does NOT cause an EasyLab program to be excluded by sequential line numbers.
- Line numbers are used as reference markers for program braching statements.

Example Program: JOOCHAN.DEMO.IFTHEN

```
➢ 1     display off
➢       get.hand.a
➢       get.from.centrifuge
➢       put.into.capper
➢ 10    if c:capped = 1 then 11
➢       cap
➢       goto 100
➢ 11    uncap
➢ 100   get.from.capper
➢       rack.2.index=1
➢       put.into.rack.2
➢       park.hand
```

4 – Loop, conditional & pausing commands

✐ **4.4- Purpose:** TIMER (x) / WAIT FOR TIMER x.

Description:

There are eight internal timers available to the operator as EasyLab Language Commands that may be used to delay system operation or time events during the running of a program. Module may operate while timers are active. More than one timer may be in use at the same time.

Format:

To set timers:

TIMER(x) = *math expression*

where *x* is equal to 1 through 4 (1 through 8 in non-PyTechnology systems) and math expression is any constant or variable representing time in seconds.

ⓘ when a program line setting a timer is encountered during the running of a program, the following ocurrs:

a) The Controller begins counting the time specified in 1 second increment.
b) The Zymate System continues operating as if this program line had never been encountered.

To wait for timers:

WAIT FOR TIMER x

where *x* is equal to the timer set – 1 through 4 or 1 through 8 in non-PyTechnology systems. “x” should NOT be enclosed in parentheses in this format.

ⓘ when the WAIT FOR TIMER statement is encountered during the running of a program, the following ocurrs:

a) The Controller checks the elapsed time of the timer specified.
b) If the time specified has elapsed, TIMER(1) = 0, system operation continues with no pause.

4 – Loop, conditional & pausing commands

c) If the time specified has NOT elapsed, TIMER(x) greater than 0:
   - c1) The module currently operating finishes its command.
   - c2) The Zymate System "pauses" until the time specified has counted down.
   - c3) The system continues from where it left off.

Remarks:

- Each timer can be set to a maximum of 65535 seconds
- Current timer values (using timer array facility) may be used in math expression as a variable.
- The setting of a timer as no effect on system operation. The modules continue to operate as if that program line was never encountered.
- Program execution is suspended when:
  The WAIT FOR statement is encountered AND
  The time specified has not elapsed.
- The timer value may be checked during the running of a program by requesting the value of the timer array.
- A timer may be set in a program or in Manual Control.
- When the Zymate System is actively waiting for a timer to elapse, any actions taken by the operator are ignored.
- If a program has to be aborted while it is waiting for a timer, it must be done by turning off the Controller's power.
- Several program statements may appear between the TIMER(x) = math expression and WAIT FOR TIMER x statements in a program. (This allows the system to continue operating while timers are active).

Example Program: JOOCHAN.PAUSE

```
➢ display off
➢ prompt Insert pause time:
➢ input joochan.pause
➢ timer(1)=joochan.pause
➢ wait for timer 1
```