DOI: 10.1051/matecconf/20165607002

Development of a Control and Vision Interface for an AR.Drone

Prasad Cheema*, Simon Luo* , and Peter Gibbens

School of AMME, The University of Sydney, 2006, Australia

** Equal contributions*

Abstract. The AR.Drone is a remote controlled quadcopter which is low cost, and readily available for consumers. Therefore it represents a simple test-bed on which control and vision research may be conducted. However, interfacing with the AR.Drone can be a challenge for new researchers as the AR.Drone's application programming interface (API) is built on low-level, bit-wise, C instructions. Therefore, this paper will demonstrate the use of an additional layer of abstraction on the AR.Drone's API via the Robot Operating System (ROS). Using ROS, the construction of a high-level graphical user interface (GUI) will be demonstrated, with the explicit aim of assisting new researchers in developing simple control and vision algorithms to interface with the AR.Drone. The GUI, formally known as the Control and Vision Interface (CVI) is currently used to research and develop computer vision, simultaneous localisation and mapping (SLAM), and path planning algorithms by a number of postgraduate and undergraduate students at the school of Aeronautical, Mechanical, and Mechatronics Engineering (AMME) in The University of Sydney.

1 Introduction

Quadcopters have recently garnered a lot of interest in the aerospace community. This is predominantly due to the simple nature of their force calculations (only lift and thrust required), and the inexpensive nature of their construction which allows for the wide spread availability of this platform [1, 2]. This has ultimately lead to the production of the AR.Drone - a cheap quadcopter freely available for purchase by the public. As a result, the AR.Drone has received much interest to act as a research platform for robotic research in the fields of control and vision processing.

Traditional quadcopter research has involved using computer vision for autonomous flight in an unmapped environment, via simultaneous localization and mapping (SLAM) [3,4,5]. In the past, many researchers have relied on expensive aerial vehicles with computationally intensive sensors to accomplish SLAM [6]. However, recent research efforts have attempted to visually map the environment using low cost drones such as the Parrot AR.Drone by using efficient algorithms [7,8]. In addition to SLAM, a pertinent example may be seen by the Computer Vision Group at Tehcnische Universitat Munchen (TUM). This is most noted in their use of the AR.Drone as a platform for control algorithms and vision navigation research, as well as a tool for educational and outreach purposes. Their research mainly pertains to the use of monocular RGB-D cameras to implement SLAM for the purpose of scale aware autonomous navigation and dense visual odometry [9]. In their research the vision group at TUM have employed a PID controller successfully on the Parrot Drone [9]. Control techniques similar to those demonstrated by TUM have been shown in Altu & Taylor [10], as well as Bristeau et al. [11]. Furthermore, as researchers at Cornell University demonstrate, a particular class of drones known as micro aerial vehicles (MAV) have been able to be autonomously move in an indoor environment using algorithms based on feature detection. More specifically, canny edge detection has been used in conjunction with a probabilistic Hough transform to find the lines of a staircase. The MAV also uses the vanishing point (the furthest point until the image becomes a point) to navigate around a particular staircase [12].

Thus, there is a variety of research performed on MAV-type drones, but with the emergence of the AR.Drone a cheap and readily available research alternative to the traditional, more expensive drone models has been developed. It has been utilised in a variety of research, but due to the complexity of these research tasks it is often the case that the hand-over phase to new researchers is unreasonably long, and has a high associated learning curve. Thus, it is the purpose of this paper to try and develop a control and vision interface (CVI) in order to assist new researchers in control and navigation field, for interfacing with the AR.Drone.

2 Overview of the AR.Drone

The AR.Drone 2.0 is a remote controlled consumer quadcopter developed by Parrot. The body is made of a carbon fibre tube structure and high resistance plastic. A protection hull is made of Expanded Polypropylene (EPP) foam which is durable, light in weight, and recyclable. The propellers are powered by four brushless motors (35 000 rpm, 15W power), and on-board energy is provided by a Lithium polymer battery with a capacity of 1000 mAh. This allows a flight time of approximately 10 minutes. The internal processor of the Drone is a 468 MHz ARM9-processor and comes included with 128 Mb of RAM memory. The processor runs a minimalist version of Linux on board. Moreover an integrated 802.11g wireless card provides network connectivity with an external computing device via WiFi.

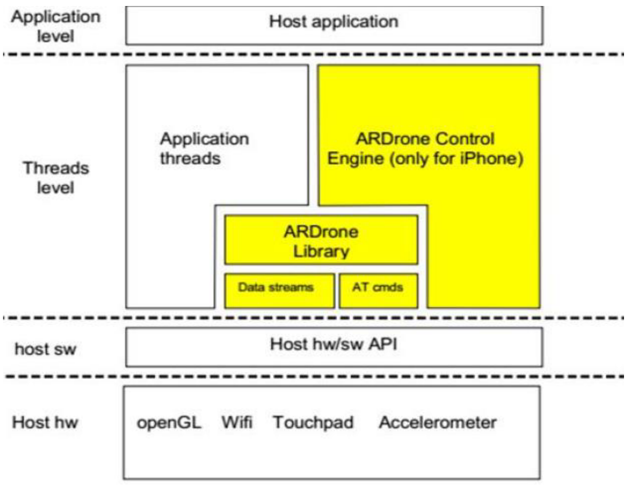

Figure 1. The layered architecture of the AR.Drone 2.0's SDK.

In regards to sensor technology and actuators, the AR.Drone 2.0 is equipped with a 3-axis accelerometer (BMA 150), and a 2-axis roll and pitch gyroscope (IDG-500) with an additional single axis gyroscope (Epson Toyocom XV-3500CB), which are of MEMS type. An ultra sound altimeter is also included for height calculations. Moreover the Drone has two cameras (forward and down facing). The forward facing camera covers a field of view 92^{o}, has a resolution 640 x 360. According to literature it generally contains significant radial distortion that needs to be corrected through via calibration techniques. The second camera points down, covers a field of view of 64^{o} and records at 60 fps. At any point in time only one video stream can be used to record video [9]. For movement the AR.Drone relies upon differential torque and thrust. By spinning each rotor with a different amount of angular velocity, the three basic motions of flight mechanics, yaw, pitch and roll can be achieved.

The Software Development Kit (SDK) that is used within AR.Drone is a low level C-based interface. Ultimately, it allows third party developers to create new software for the AR.Drone through the use of libraries which aid in vision processing and controllability of the drone. However, it must be noted that the SDK does not support the writing of one's own embedded software. That is to say, it offers no direct access to manipulate low level drone hardware. Figure 1 shows a layered architecture overview of the AR.Drone SDK. The yellow coloured portions of the diagram are those parts which the user has direct control over (for example the programmer will have the ability to read in data streams, as well as send data streams). Note that the AR.Drone Control Engine exists only specifically for the iOS devices. This allows the AR.Drone's Application Programming Interface (API) to be portable for such devices.

Table 1. Communication port streams used by the AR.Drone.

Port Number	Communication Type	Function
5554	UDP	Navdata stream sent from Drone to client
5555	TCP/IP	Image stream sent from Drone to client
5556	UDP	Controlling and configuring the Drone using commands
5559	TCP/IP	Transferring of critical system information

The SDK communicates through to an external PC via UDP and TCP connections for different purposes. These communication ports are outlined in Table 1.

Although very complete, the main drawback to the use of the SDK system is its low level nature. Instead of sending direct commands such as "move forward with a particular speed", it is necessary to send a complete bit sequence, with sequences using an unsigned 32 bit representation. This means that sending a complex command sequence to the drone can become very cumbersome. Moreover, working with the drone on a complex level, such in the case of SLAM, will involve an incredibly large amount of software development for even the simplest of SLAM algorithms to work. This is definitely counter-productive for the purpose of engineering research. If students are required to perform and learn large amounts of low level software programming it will slow down and perhaps even limit efforts in the use of drones for future research projects.

Hence, it was decided to implement the communication to the AR.Drone via the Robot Operating System (ROS) ARDrone autonomy driver. This driver provides a higher level abstraction to the SDK via the ROS system. It is still relatively low level in that it deals with Drone via C++, but the abstraction it provides allows the AR.Drone to move forward, for example, through a single velocity command rather than building up multiple bit streams. Of course understanding the manner in which ROS works to communicate with the drone does have an associated learning curve, however ROS is a well-documented and widely used environment for robotic research development that gaining familiarly for a ROS environment will be an ineffably easier task compared to learning to work with the even lower level (and poorly documented) SDK. A look into the ROS environment is explored in the following section.

3 Development of the Control and Vision Interface

This section of the paper explores the development of the control and vision interface (CVI) as a whole. It will first look at the C++ systems use build the CVI (through ROS and OpenCV). From there it shall explore how these components work together to generate a coherent computer system for researchers.

3.1. Understanding Robot Operating System

The Robot Operating System (ROS) is an open source platform used to aid communication with research robot. It is a `so called' operating system because it provides similar services to an OS. For example it includes multi-levelled hardware abstraction, low level device control, inter-process communications, and distributed package management. In addition to this, it is especially venerated within the robotics community due to its ample tools and libraries of algorithms, which allows for multi-layered communication with robots. There are several distinct advantages to the uses of ROS, which are briefly explored in the following sub-sections [13].

3.1.1 Distributed computation

ROS provides an abstraction system to help simplify distributed computations on robots. Distributed computation naturally arises throughout the use of multiple computers within the single robot, or when multiple robots attempt to cooperate and coordinate their efforts to achieve some overarching goal. Moreover, it is simpler to subdivide one's code into multiple blocks which can then co-operate together via inter-process communication (as is used in the development of the CVI's control and vision GUI).

3.1.2 A single, monolithic library

Although ROS contains some core libraries, more often than not the user will be required to download additional libraries to supplement their current library (or to use external code developed by other universities). The compilation and linkage of these libraries can cause significant issue if the user is not well acquainted with such aspects of software development. Moreover the linking of several external libraries will generally exponentially increase compilation time for the program.

3.2 Understanding OpenCV

In the construction of the CVI, the open computer vision (OpenCV) library was used for image processing. Briefly, OpenCV is an open source computer vision project supplied online by sourceforge. All libraries and systems of OpenCV are written in C and C++, and projects from OpenCV are able to run under Linux, Windows and the Mac OS X operating systems. Recently, there have been substantial efforts to port the OpenCV libraries to Python, Ruby, and MATLAB, although progression is generally slow, and support is very little. Due to OpenCV's dependency on the low level languages of C and C++, a strong focus of the OpenCV libraries is on computational efficiency and real time applications. It does this by looking at specific memory allocations on the processor level, and can in most cases utilize multi-core processing (a strong limitation of MATLAB for real time computer vision is that it can only use one core at most, unless an external parallel computing toolbox is installed). The main goal of OpenCV is to provide a simple-to-use computer vision infrastructure which will help people construct fairly sophisticated computer vision processes rapidly.

3.3 Development of Control and Vision Interface

Speaking from a high level perspective, the CVI development required a large degree of internal multi-threading. This is necessary because both of these applications (the control part - Control GUI, and the vision part - Vision GUI) require multiple infinite loops in order to function properly. For example a standard GUI thread will always require an infinite loop to physically show itself on the screen, and to perform checks on the buttons. The running of ROS would need its own infinite loop to check the global call back queue defined in the ROS API, and finally the "Video Thread" and "Video Processing Thread" both require their own threads. This is because the showing of the camera GUI showing, the video stream (and its processed version of itself) all require infinite loops in order to continue to compute image data and show it on screen. Thus the GUIs are in fact highly threaded to internalise computations and render them quick enough for a psuedo-real time environment to be used. A summary of this multi-threaded nature is shown in Figure 2.

In addition to multi-threaded code, the two GUIs (and a third process labelled as pseudo real-time graph which graphs state vector outputs) are able to communicate through Inter Process Communications (IPCs). However IPC is not the ideal option to communicate data between each GUI. If all the GUIs were internally multi-threaded into one another, then that would be most efficient. However threading all aspects into a single GUI would make the coding extremely difficult (lots of inherent multi-threading issues such as context switching and race conditions will arise), and therefore difficult to conceptually follow for new researchers in the field. Thus two (separate) processes were selected as the time optimal way to code up these applications.

Ultimately, the control GUI aims to facilitate control-based communication with the AR.Drone. It achieves this through the use of keyboard control, open access to navigation data, and button options to perform a pseudo real time plot of system states. In addition it provides the user with a link to a second GUI for video processing. The final, coded implementation of the control GUI is diagrammatically represented in Figure 3.

In Figure 3 it can be seen the user has the option to manually control the flight of the AR.Drone through simple keyboard controls as well as track the state variables defined in Table 2 in pseudo real-time.

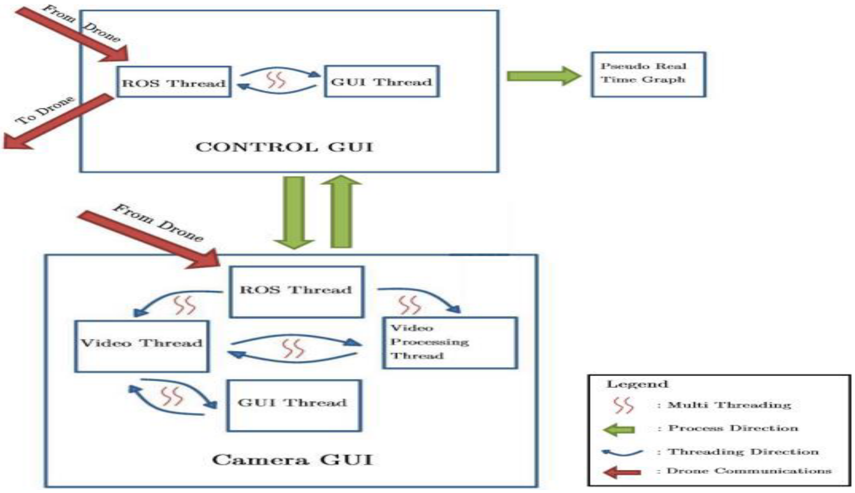

Figure 2. Demonstration of the Multi-Threaded Nature of the CVI back end.

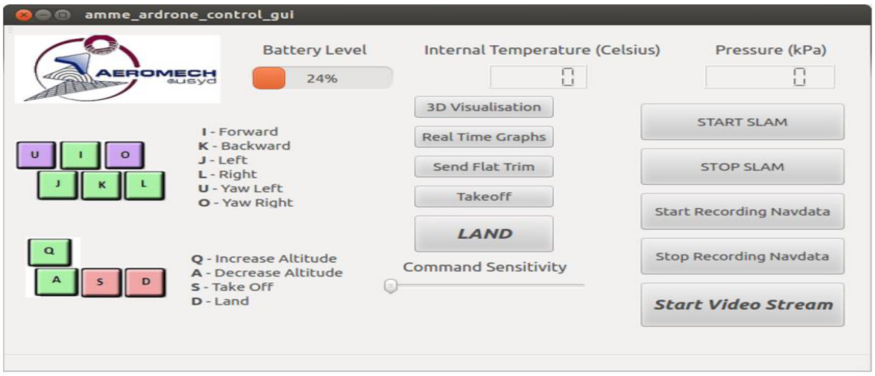

Figure 3. Final implementation of control GUI in its 'off' state.

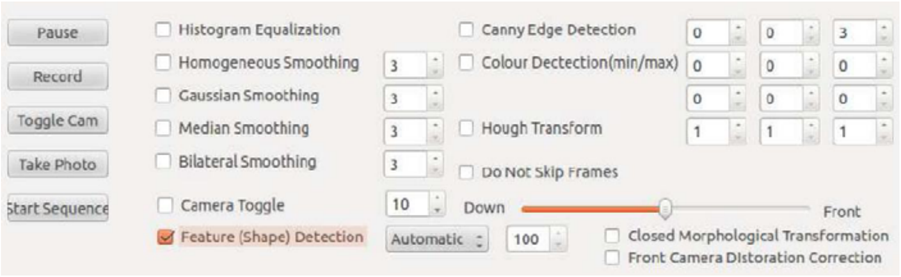

Figure 4. Final implementation of Camera GUI showcasing pseudo real-time edge detection.

Table 2. Summary of AR.Drone information and corresponding units extracted from the SDK via ROS.

Variable Recorded	**Units**
Timestamp	Second since Unix Epoch
Time between timestamps	Nanosecond
Rotation rates	Radians/second
Acceleration	Metres/second^2
Quaternions	Unitless
Euler Angles	Degrees
Altitude	Centimetres
Stream sequence number	Unitless

In addition to the ability to perform and track pseudo real-time control operations on the AR.Drone, a link can be directly set-up to the video steam GUI which can be initiated by clicking the *Start Video Stream* option shown in Figure 3. The completed state of the video GUI portion of the CVI interface is represented in Figure 4.

From Figure 4, it can be seen that the user is given the option to toggle between the front, and down facing cameras for the video stream. This is a consequence of the AR.Drone 2.0 only being able to show one video stream at one point in time via the ROS driver. Moreover, the GUI enables researchers to toggle between various image filtering techniques in pseudo real-time which is an invaluable experience for learning, as well as for research. In particular, Figure 4 shows an implementation of an in-house contour-based feature detection algorithm.

Although the CVI shown in Figures 3 and 4 seem extensive, obviously there are a multitude of other algorithms which can be implemented for various research purposes. Therefore, in order to accommodate the possibility of future research the code has been constructed in a modularised fashion, so that it essentially acts as API. Therefore future researchers need only use a few basic functions (the same ones used to create all the other buttons and features) within which new code can be written and dynamically integrated into the CVI. A few examples of the positive impacts of the CVI for future drone-based research is outlined in the following sections.

4 Application to Real Time Feature Detection

The framework for the Vision Node GUI has been adapted such that additional functions can be easily be added or removed into the video stream. The vision GUI has been extended to include real time feature detection. This is done by adding a feature detection algorithm to the video processing thread as shown in Figure 3. This algorithm includes a number of stages, first a Gaussian smoothing filter is applied to remove all the background noise. This is followed by Canny edge detection, which creates a binary image, which in turn places a pixel at regions with a large gradient change. This is then followed by the Suzuki algorithm [14] which is used to find closed contours in the image. These algorithms have been implemented in the CVI framework. The results are shown in sub figures 5 (a) and 5 (b).

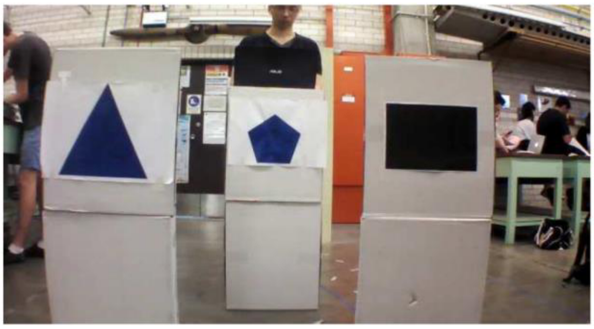

Figure 5(a). The raw image from the AR.Drone.

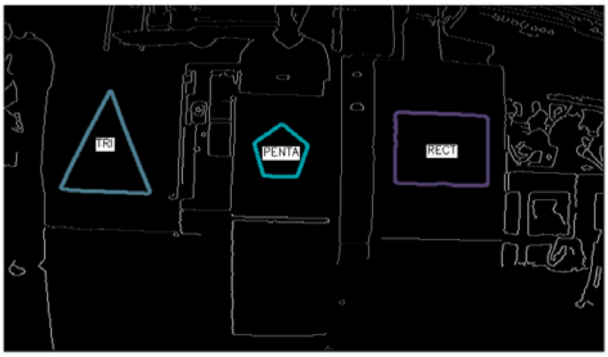

Figure 5(b). The processed image with edge detection.

It can be seen in Figure 5 (b) that the feature detection algorithm converted the raw image into a binary image which consists of only the edges of the objects in the images. From there the aforementioned Suzuki algorithm detects a closed contour on the image. These closed contours are then matched against a database of features, in this case a triangle, rectangle and a pentagon. By implementing this multi-stage feature detection algorithm, it can be seen that the framework is robust enough to be able to support multi-stage algorithms.

Hence, the vision GUI has developed a pipeline which is highly effective in adding and removing computer vision algorithms. Even with algorithms which require high computational power such as Canny edge detection and contour detection algorithms. The framework is set up such that each critical section is isolated in its own thread allowing the software to run smoothly. When running in a computer with lower computational power, the frame work ensures that the video stream is running in real time by only retrieving the newest video frame.

5 Application to Monocular SLAM

Following the success of the feature detection, the CVI was extended to incorporate a SLAM framework. SLAM is a computational problem which requires user to develop a map of an unknown environment whilst keeping track of its location. The CVI has assisted in the development of SLAM by providing a simple API for mapping, control and video stream. The overall mapping and feature detection can be seen in Figure 6. The mapping feature uses the 3D visualisation tool Rviz provided by ROS. The feature detection algorithm is run in parallel with the SLAM algorithm in the vision node. The full SLAM algorithm has been implemented in the control node, with the features and landmarks detected through the vision GUI.

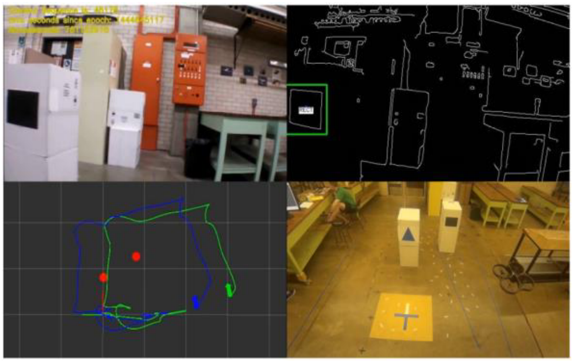

Figure 6. Map building and feature detection working for SLAM.

Ultimately, the effective use of multi-threading has allowed the control of the drone and SLAM algorithm to run simultaneously in the control node. Figure 6 shows that SLAM has been effectively implemented into the CVI framework. The mapping of the features during SLAM can be seen towards the bottom left corner of Figure 6, where the red dots indicate the features, green line showing the dead reckoning and the blue line indicating the prediction by SLAM. The implementation of SLAM has demonstrated an effective communication between the control and vision node and as well as the highly effective usage of using threads to ensure that all features in the CVI run in pseudo real-time.

6 Conclusion

In conclusion this paper has demonstrated the development and application of a control and vision interface (CVI) for use on a commercial quadcopter. Using the AR.Drone 2.0 as a simple, low-cost, readily available test-bed it is possible to research into a wide-range of control and vision-based applications. This paper has related how the use of a high level abstraction via OpenCV and ROS work intimately together to create a coherent and sufficiently abstracted interface to assist in user interaction with the AR.Drone. Ultimately this paper serves as an important introductory platform for any researchers new to the field of vision processing or controller design for quadcopter systems, as the basic structure and methodology behind interfacing with the quadcopter is explored here. Examples of the practicality of this approach have been demonstrated in the fields of real time feature detection and monocular SLAM.

Acknowledgements

The authors would like to thank the undergraduate students Andy Ho, and Trevor Wang, of The University of Sydney. They were able to assist in the appropriation of the CVI for the SLAM and feature detection applications explored in this paper.

References

1. M. Saska, T. Krajnik, J. Faigl, V. Vonasek, L. Preucil, IEEE IRS, 4808 – 4809 (2012)
2. M. Michael, The AR Drone LabVIEW Toolkit : A Software Framework for the Control of Low-Cost Quadrotor Aerial Robots (Tufts Univ. Ph. D. 2012)
3. S. Yang, S. Scherer, and A. Zell, IEEE ICRA, 5227 – 5232 (2014)
4. F. Sadat, S. A., K. Chutskoff, D. Jungic, J. Wawerla J., R. Vaughan, IEEE ICRA, 3870 – 3875 (2014)
5. G. H. Lee, F. Fraundorfer, and M. Pollefeys, IEEE ICRA, 3139 – 3144 (2011)
6. N. Karlsson, E. di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. E. Munich, IEEE ICRA, (2005)
7. N. Dijkshroon, Simultaneous Localisation and Mapping with AR.Drone (Amsterdam Univ. Ph. D. 2012)
8. M. Blosch, S. Weiss, D. Scaramuzza, and R. Siegwart, IEEE ICRA, (2010)
9. J. Engel, J. Sturm, D. Cremers, J. of Rob. And Auto. Systems **62**, 1646 – 1656 (2014)
10. E. Altug, C. Taylor, IEEE ICM, 316 – 321 (2004)
11. P. J. Bristeau, F. Callou, D. Vissiere, and N. Petit, IFAC World Congress, 1477 – 1484 (2011)
12. C. Bills, J. Chen, and A. Saxena, IEEE ICRA, 5776 – 5783 (2011)
13. J. M. O'Kane, A Gentle Introduction to ROS, (Tech. Rep. 2014)
14. S. Suzuki, J. of Comp. Vis., Graphics, & Img. Proc. **30**, 32 – 46 (1985)