UNIVERSITA' DEGLI STUDI DI PADOVA

Corso di Laurea Magistrale in Ingegneria dell'Automazione

Tesi di laurea

Sistema di Monitoraggio da Remoto per Imbarcazioni: Progetto ed Implementazione

Tesista: Mirko Perazzolo
Relatore: Prof. Stefano Vitturi

Anno Accademico 2015-2016

Indice

Sommario

eKEEPER2 è un progetto imprenditoriale che si vuole avviare nel settore della strumentazione per le unità da diporto nautico. L'idea di business prevede la realizzazione e vendita di uno strumento per il controllo di alcune funzioni e stati da remoto delle imbarcazioni da diporto, con un approccio al cliente di tipo non convenzionale, in quanto fa leva sul desiderio dell'armatore di interagire con la propria imbarcazione anche quando vi è fisicamente lontano.
Il prodotto è caratterizzato dalla possibilità di controllare da remoto, mediante smartphone, alcune utenze tipiche delle imbarcazioni, ad esempio richiedere ed ottenere informazioni relative a parametri tipo localizzazione della barca, stato delle batterie, presenza di alimentazione elettrica da rete, temperatura ed umidità, oppure ricevere allarmi in caso di situazioni anomale come il superamento del livello di soglia dell'acqua in sentina, spostamento della barca dalla posizione di riferimento o di intrusione. Lo studio del prototipo qui presentato si articola in tre distinti punti:

1. Individuazione dei componenti elettronici per la realizzazione del prototipo, e relativa programmazione
2. Creazione di un'applicazione per Smartphone con sistema operativo Android
3. Allestimento di un server al fine di permettere la comunicazione tra eKEEPER2 e smartphone via internet

Verranno presentati gli aspetti fondamentali dell'implementazione di ognuno dei tre punti, affiancando ad essi i risultati ottenuti dai test sul prodotto ultimato.

Si ringrazia Aitech s.r.l. per la gentile collaborazione nello sviluppo del progetto e per la possibilità di crescita offerta.

1 Introduzione

L'obbiettivo che l'azienda Aitech s.r.l si pone è quello di realizzare uno strumento utile al cliente diportista nautico per monitorare la propria imbarcazione da remoto, in qualsiasi istante e da ogni luogo in cui si trovi.
L'interfaccia tra strumento e cliente sarà lo smartphone, dispositivo estremamente diffuso, il quale sfrutterà la comunicazione via SMS e/o via internet per mezzo di un'applicazione appositamente studiata. Lo strumento andrà ad integrare la normale strumentazione di bordo, senza sostituirsi ad essa, fornendo un servizio di controllo da remoto con la migliore tecnologia disponibile.
In fase di progettazione sono stati intrapresi i seguenti step:

- Selezione di componenti e fornitori adatti allo scopo;
- Realizzazione di un prototipo dello strumento da collaudare e certificare prima del lancio nel mercato del prodotto finito;
- Implementazione del software per il funzionamento dello strumento e per la gestione dei prodotti venduti;
- Realizzazione dell'applicazione per smartphone.

La missione di Aitech s.r.l., pertanto, è fare di eKEEPER2 un punto di riferimento per chi vuole integrare la propria strumentazione di bordo con un sistema di controllo da remoto che permetta di tenere attivo un filo diretto con la propria barca. Al fine di poter proporre diverse fasce di prezzo, quanto sopra si concretizza realizzando un prodotto base personalizzabile a seconda dei desideri dell'armatore che potrà scegliere il prodotto finale da ordinare aggiungendo altri moduli che permetteranno di ottenere le funzioni che desidera.

L'area che Aitech s.r.l. ritiene di poter servire è per il momento circoscritta all'Italia, per estendersi in fasi successive anche alle zone costiere delle nazioni bagnate dall'Adriatico e dal Tirreno, secondo una logica di estensione progressiva del mercato.
Il cliente tipo è il possessore di una unità da diporto di lunghezza tra i 10 e i 18 metri, nel caso di unità a vela, con limite inferiore estendibile agli 8 metri nel caso di unità a motore. In ogni caso si concentrerà lo studio sui cabinati ormeggiati nelle strutture italiane.
Basandosi sul rapporto UCINA 2014 (Confindustria Nautica), su un totale di 409 marine e porticcioli, in Italia sono disponibili 156.606 posti barca: di questi il 10% deve essere per normativa lasciato libero per i transiti, quindi si può considerare un totale di 140.945 posti barca disponibili che vengono ridotti di un ulteriore 10% relativo ai posti barca non venduti (per effetto della crisi/migrazione) ottenendo un totale di barche ormeggiate pari a 126.850.
Di tutte le barche ormeggiate si può considerare che una parte sia del tipo cabinato e le altre siano dei day cruiser che quindi in linea di massima non necessitano della strumento proposto.
Nel complesso, si ritiene di non sbagliare di molto se si considera come mercato potenziale di riferimento le 104.000 unità immatricolate (per unità di lunghezza maggiore di 10 m è obbligatoria l'immatricolazione) che corrisponde al 66.4% dei posti barca disponibili in Italia.
Si può riassumere quanto sopra osservando la Tabella 1:

N. posti barca in Italia		156.606
Posti liberi transito	10%	15.661
Posti occupati	81%	126.851
Mercato Potenziale	66.4%	104.000

Tabella 1: Mercato potenziale

2 Analisi del progetto

Il progetto presentato nel seguente documento nasce da un dispositivo già presente nel mercato.

Aitech ritiene che tale dispositivo debba tuttavia costare meno al cliente e quindi siano necessarie modifiche in particolare per quanto riguarda i costi di produzione.

Il punto di partenza è quindi questo modello da considerarsi ormai datato e del quale non si conoscono in dettaglio le caratteristiche progettuali, in quanto di proprietà del precedente fornitore.

2.1 Analisi del dispositivo

Il primo passo riguarda un lavoro di *Reverse Engineering* sul dispositivo esistente.
Le conoscenze a priori sul dispositivo si basano sull'esperienza maturata come utilizzatore da parte di Aitech, e sulle istruzioni che vengono fornite in allegato allo stesso.
Seguendo tale analisi, è stato possibile determinare gli ingressi e le uscite utilizzate e delineare gli stati di funzionamento del dispositivo.

Figura 1: Modello precedente del dispositivo sviluppato

2.1.1 Ingressi e uscite

Una prima analisi riguarda l'osservazione del dispositivo dalla quale si può ottenere un'informazione sull'interfaccia hardware del quale è dotato. Si riconosce facilmente in Figura 1 che il dispositivo monta due pannelli principali dove sono posizionati connettori ed eventuali alloggiamenti. Arricchendo l'informazione visiva che si è così ottenuta con l'informazione contenuta nel manuale di montaggio, si possono riconoscere i singoli connettori e le relative funzioni alle quali sono adibiti.
Si riportano di seguito i risultati di questa analisi:

- Connettore di alimentazione

- Connettore per la comunicazione I^2C
- Connettore per il collegamento degli ingressi e le uscite ai seguenti dispositivi:
  - Sensore di acqua in sentina (ingresso)
  - Sensore di apertura della porta di accesso alla sottocoperta (ingresso)
  - Sensore di attivazione dell'interruttore staccabatteria (ingresso)
  - Pulsante ausiliario per richiesta assistenza (ingresso)
  - Relè dispositivo 1 (uscita)
  - Relè dispositivo 2 (uscita)
  - Relè dispositivo 3 (uscita)
- Connettore per la comunicazione RS232
- Connettore per l'antenna GSM
- Connettore per l'antenna GPS
- Slot per l'inserimento della scheda SIM
- Indicatore luminoso (LED) per segnalazione di corretto funzionamento

Con solo questa prima analisi è possibile delineare a grandi linee che tipo di funzionalità implementa questo dispositivo. Infatti dalla presenza di un alloggiamento per una scheda SIM e dell'antenna GSM e dalla mancanza di un'interfaccia *user-friendly* si intuisce che il principale metodo di comunicazione utilizzato sia attraverso la rete cellulare.
Inoltre ha la possibilità di determinare la propria posizione, in quanto prevede un'antenna GPS.
È anche evidente che il dispositivo sia in grado di ottenere informazioni da quattro sensori e controllare tre dispositivi.
Per quanto riguarda i connettori per le comunicazioni I^2C e RS232, i *pinout* utilizzati sono presenti nel manuale. Nonostante questo, il fatto che siano stati utilizzati due connettori non standard per tali applicazioni e non vi siano informazioni sui dispositivi e sulle comunicazioni stesse a loro associati, premette di dedurre che siano riservati ad operatori autorizzati, quindi eliminabili nel caso si trovino altre vie per eventuali analisi in caso di guasto.

2.1.2 Funzionalità

Si analizzano ora le funzionalità implementate nel dispositivo che lo rendono un'ottima soluzione per il monitoraggio e il controllo da remoto dell'imbarcazione.
In questa analisi si sono esaminati gli elementi sui quali si basa la pubblicità del dispositivo e su quanto descritto nel manuale dello stesso.
Quello che si è ottenuto è riportato di seguito:

- La rilevazione e segnalazione dei seguenti eventi anomali:
  - Presenza di acqua nella sentina
  - Apertura della porta di accesso alla sottocoperta
  - Attivazione dello staccabatteria
  - Pressione del pulsante ausiliario di assistenza
  - Spostamento dell'imbarcazione all'infuori della zona di sicurezza

- Il controllo remoto di tre dispositivi presenti nell'imbarcazione (quali calorifero, frigorifero, pompa della sentina, etc)
- La registrazione dei viaggi compiuti
- Avvisi e comunicazioni via SMS

Questa analisi risulta essere concorde con quanto ottenuto nell'analisi precedente, e va ad arricchirne le informazioni. Infatti si è ora a conoscenza del fatto che il dispositivo, oltre ad essere in grado di leggere dai sensori, riconosce l'informazione e nel caso di eventi anomali provvede ad effettuare una segnalazione via SMS.
Allo stesso modo, si osserva che oltre ad essere in grado di controllare tre dispositivi, permette al cliente di controllarli da remoto via SMS.
Infine si hanno informazioni sull'utilizzo che ne viene fatto dell'informazione che si ottiene dalla connessione GPS, ovvero registrazione dei viaggi e segnalazione di spostamenti non autorizzati. La funzione di registrazione dei viaggi, suggerisce che il dispositivo disponga anche di una memoria sufficientemente capiente da registrare un numero non trascurabile di valori.

2.1.3 Stati di funzionamento

L'analisi fin qui compiuta non è ancora sufficiente a delineare il dispositivo. Infatti, come viene descritto nel manuale, Ekeeper non si comporta sempre allo stesso modo, ma ha differenti tipi di funzionamento selezionabili, che permettono di adattarne il comportamento alle esigenze del cliente, senza dover accendere o spegnere fisicamente Ekeeper.

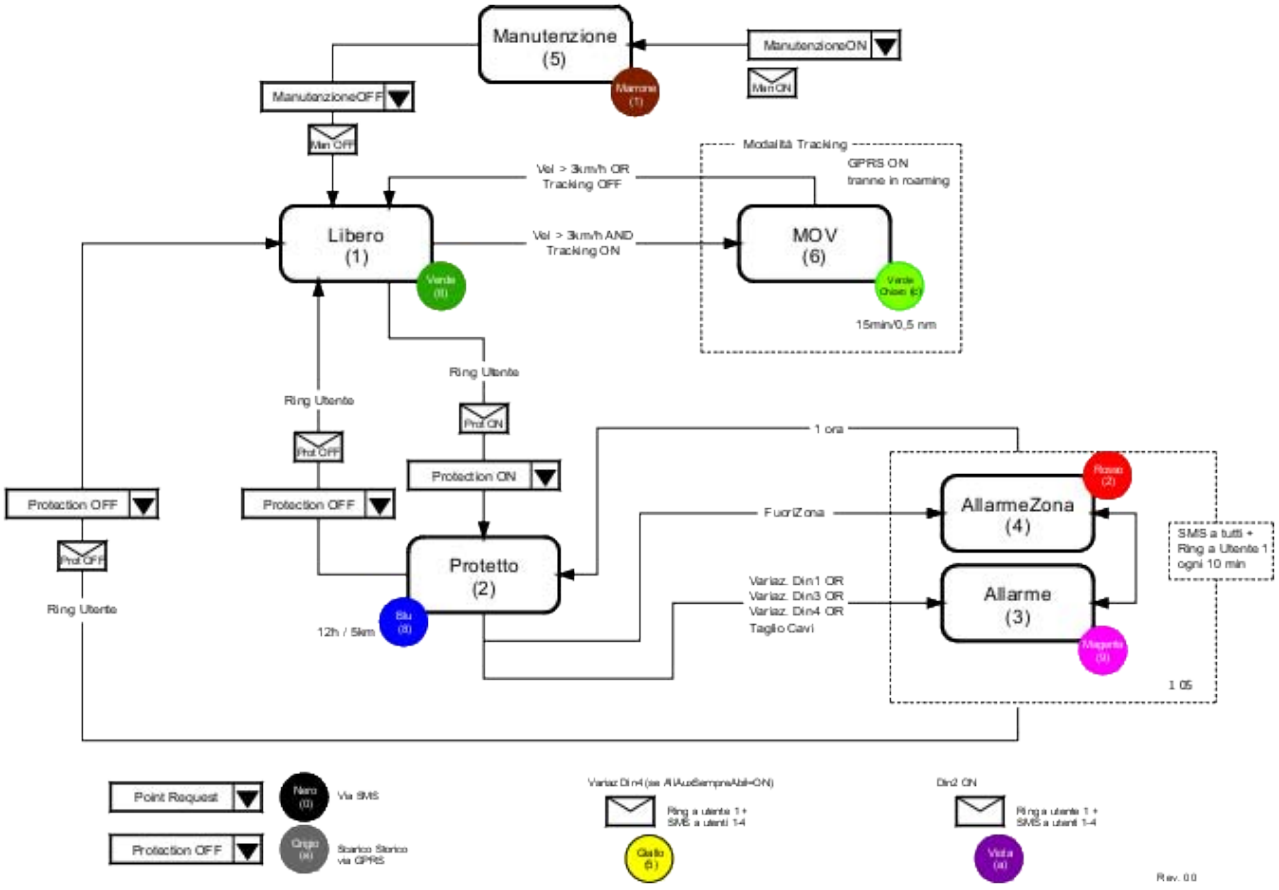


Figura 2: Diagramma di funzionamento riportato nel manuale utente del modello precedente.

Il funzionamento del dispositivo, per come traspare all'utilizzatore, può essere riassunto nel diagramma riportato in Figura 2.

Come già detto, ogni stato di funzionamento ha funzionalità specifiche, adatte ai differenti momenti di utilizzo dell'imbarcazione. Se ne riporta quindi una breve descrizione:

1. **Libero** Lo stato Libero è lo stato di funzionamento normale. In questo stato viene memorizzato lo stato dell'imbarcazione ogni 12 h o, se l'imbarcazione si sposta, uno ogni 2.7 miglia nautiche.

2. **Protetto** Lo stato di Protetto corrisponde all'attivazione della modalità antifurto. In tale stato viene avviata la funzione di monitoraggio degli ingressi e il controllo fuori-zona. In questo stato viene memorizzato lo stato dell'imbarcazione ogni 12 h.

3. **Allarme** Nello stato di Allarme il sistema

   - invia una chiamata telefonica all'utente allertabile principale;
   - invia un SMS verso tutti gli utenti allertabili con indicazioni della causa dell'allarme.

   Queste operazioni vengono ripetute ogni 10 minuti per un'ora dal primo evento di allarme, a meno che non venga disattivato. Ogni eventuale altra attivazione degli ingressi di allarme viene segnalata agli utenti, una sola volta, via SMS e con lo squillo all'utente principale.

4. **AllarmeZona** Nello stato di Allarme Fuori Zona il sistema

   - invia una chiamata all'utente allertabile principale;
   - invia un SMS verso tutti gli utenti allertabili con indicazioni della causa dell'allarme.

   Queste operazioni vengono ripetute ogni 10 minuti per un'ora dal primo evento di allarme, a meno che non venga disattivato. Ogni eventuale altra attivazione degli ingressi di allarme viene segnalata agli utenti, una sola volta, via SMS e con lo squillo all'utente principale.

5. **Manutenzione** Il sistema si pone nello stato di Manutenzione. Questo funzionamento è adatto nel caso l'imbarcazione sia in cantiere, quindi soggetta a spostamenti giustificati all'interno dello stesso. Non vengono eseguite altre azioni (registrazione dello stato, segnalazioni di eventi anomali, etc.).

6. **Mov** Nello stato Mov il dispositivo registra la posizione ogni 15 minuti o 0.5 miglia nautiche.

2.2 Analisi delle funzioni aggiuntive e dei vincoli progettuali

In questo caso, l'analisi si concentra sull'esperienza maturata come utilizzatore da parte di Aitech.
Infatti, nel riproporre questo prodotto, Aitech vuole aggiungere delle funzionalità per commercializzarlo in un mercato più ampio.

A tal fine si sono quindi studiate delle funzionalità aggiuntive che Aitech ha ritenuto essere utili per un qualsiasi cliente che intenda montare il dispositivo sulla propria imbarcazione.

Si riportano di seguito le funzionalità che sono emerse da questo studio:

- La rilevazione dei seguenti eventi anomali:
  - Assenza della tensione di 230 V della banchina.
  - Tensione della batteria servizi inferiore a tensione di soglia prestabilita
- La registrazione di dati riguardanti temperatura ed umidità interni all'imbarcazione
- Monitoraggio delle tensioni delle batterie Servizi e Motore dell'imbarcazione e della batteria tampone per garantire il funzionamento anche nel caso venga meno l'alimentazione
- Avvisi e comunicazioni via connessione internet
- Utilizzo di un'applicazione per dispositivi Android e iOS per l'interfaccia utente

L'analisi fin qui presentata ha permesso di definire la struttura che dovrà avere il nuovo dispositivo.
Prima di procedere con la progettazione del prototipo, Aitech ha ritenuto utile porre dei vincoli progettuali. Tali vincoli sono nati in seguito ad un'attenta analisi e sono stati ritenuti necessari al fine di poter entrare nel mercato con il nuovo dispositivo.
I riferimenti che Aitech ha richiesto di rispettare durante tutta la fase di progettazione e sviluppo del dispositivo sono riportati di seguito:

- Mantenere le principali funzionalità del modello già esistente
- Implementare un metodo di comunicazione più efficace e *user-friendly* rispetto a quello già esistente
- Implementare le nuove funzionalità
- Porre attenzione ai costi per una maggior competitività nel mercato
- Utilizzare uno dei microcontrollori della gamma Arduino

3 Progettazione della struttura di funzionamento

Sulla base delle analisi compiute, si possono quindi delineare le differenti funzionalità che dovranno essere implementate nel nuovo dispositivo.

3.1 Struttura di funzionamento principale

Si definisce la struttura principale sulla quale si baserà il funzionamento del dispositivo.
Come richiesto da Aitech, seguirà quanto già visto durante l'analisi del vecchio modello, arricchendolo delle nuove funzionalità. Si è mantenuta la possibilità da parte del cliente di selezionare il tipo di funzionamento in base alle esigenze, si è quindi definita una suddivisione in stati di funzionamento che descrivono al meglio le varie occasioni di utilizzo.
Di seguito vengono descritti gli stati che sono implementati nel nuovo dispositivo:

1. **Funzionamento Libero**: Stato di funzionamento dove sono disabilitate le notifiche degli allarmi (fatta eccezione per il livello dell'acqua in sentina), permettendo comunque il controllo remoto dei dispositivi. Questo funzionamento è adatto per il normale utilizzo dell'imbarcazione, ovvero quando il diportista è nei pressi dell'imbarcazione o non ha necessità di monitorare il verificarsi di eventi considerati anomali. L'imbarcazione viene comunque monitorata registrandone lo stato una volta ogni 12 h o una ogni 2.5 miglia nautiche. Si è scelto un valore di tempo così ampio, perché in una situazione normale di giacenza dell'imbarcazione, la modifica dello stato risulta particolarmente lenta, soprattutto in situazioni normali. Stessa cosa vale per la distanza, infatti in mare con distanze tali si possono ricostruire spostamenti rilevanti. Oltre a queste registrazioni basate sul tempo trascorso o sulla distanza percorsa, si è previsto di registrare lo stato dell'imbarcazione ogniqualvolta venga effettuata una azione da parte del dispositivo stesso, quali la modifica di un'impostazione piuttosto che il controllo di un dispositivo. Questo per permettere di avere uno storico di quanto avviene al fine di agevolare il debug e l'assistenza per il dispositivo stesso.

2. **Funzionamento Protetto**: Stato di funzionamento dove sono abilitate le notifiche degli allarmi e il controllo remoto dei dispositivi. Questo funzionamento è adatto per quando il cliente lascia la propria imbarcazione ormeggiata in darsena, allontanandosi. Infatti questo stato mantiene un continuo monitoraggio dell'imbarcazione. Al presentarsi di un qualsiasi evento anomalo passa subito al funzionamento di allarme collegato al tipo di evento anomalo rilevato. Oltre al monitoraggio degli eventi anomali, si garantisce un monitoraggio dello stato dell'imbarcazione, registrandolo una volta ogni 12 h, come previsto per il funzionamento Libero. Infatti, salvo casi straordinari, i quali rientrano negli eventi anomali, si presume che l'imbarcazione non subisca alterazioni significative in tempi ristretti. Per quanto riguarda la registrazione in base alla distanza percorsa, in questo funzionamento non viene eseguita, ma viene interpretata come un evento anomalo. Infatti, viene inteso come uno spostamento non autorizzato, facendo così passare al funzionamento Allarme Zona. Infine, come avviene nel funzionamento Libero, anche in questo caso si esegue una registrazione dello stato ogniqualvolta venga effettuata una azione da parte del dispositivo.

3. **Funzionamento Allarme**: Stato di funzionamento dove vengono notificati gli allarmi riscontrati. In questo funzionamento il dispositivo provvede ad avviare la sequenza di notifica degli eventi anomali. La sequenza prevede una telefonata di breve durata al primo numero memorizzato e l'invio di un SMS ad ogni numero memorizzato dove viene indicato l'evento anomalo rilevato. Il tutto viene ripetuto ogni 10 minuti, per un massimo

di sei volte, a meno che uno dei numeri memorizzati non provveda a disattivare l'allarme mediante SMS. Questa sequenza subisce una modifica nel caso vengano rilevati ulteriori eventi anomali. In tal caso, viene immediatamente rieseguita la sequenza notificando anche i nuovi eventi anomali. Nel caso in cui tra gli eventi anomali vi sia anche l'evento "Fuori Zona" (trattato al punto successivo), il dispositivo passa nel funzionamento Allarme Zona.

4. **Funzionamento Allarme Zona**: Stato di funzionamento dove viene notificato l'allarme "Fuori zona" ed eventuali altri allarmi riscontrati. L'evento anomalo identificato come "Fuori zona" viene rilevato dal dispositivo nel momento in cui l'imbarcazione viene spostata ad una distanza pari o superiore a 1.6 miglia nautiche dalla posizione in cui si trovava l'imbarcazione quando è stato inserito il "Funzionamento Protetto". In questo funzionamento, come nel funzionamento Allarme, viene avviata la sequenza di notifica composta dall'avvio di una telefonata di breve durata al primo numero memorizzato e dall'invio di un SMS a tutti i numeri memorizzati, il tutto una volta ogni 10 minuti per un massimo di sei volte. Eventuali altri eventi anomali vengono segnalati immediatamente con l'esecuzione di un'altra sequenza di notifica.

5. **Funzionamento Manutenzione**: Stato di funzionamento dove sono disabilitate le notifiche di tutti gli allarmi. Questo funzionamento è adatto per le occasioni in cui la barca viene portata in ricovero per esempio per effettuare un'operazione di manutenzione, dove si prevedono degli spostamenti e delle modifiche all'imbarcazione che possono essere considerati autorizzati e quindi per le quali non vi sia necessità di monitoraggio. Non si è prevista una memorizzazione dello stato dell'imbarcazione temporizzata, in quanto si prevede che l'imbarcazione sia controllata e quindi risulterebbe superfluo, viene comunque registrato lo stato ogniqualvolta venga effettuata una azione da parte del dispositivo.

6. **Funzionamento Movimento**: Stato di funzionamento dove viene registrato il viaggio. Questo funzionamento (attivabile solo quando è abilitata la funzione di tracking) è adatto per una registrazione accurata del tragitto compiuto, infatti si passa ad una acquisizione ogni 1 miglio nautico o una ogni 15 minuti, molto più accurata rispetto a quanto fa il dispositivo in funzionamento Libero. Il doppio parametro (metrico e temporale) è ritenuto necessario in quanto si considera 1 miglio nautico una misura sufficientemente accurata per il tracciamento della rotta seguita, e vi è una notevole differenza tra le velocità di crociera tenute da una barca a vela ed una a motore. Se infatti per una barca a vela si può ipotizzare una velocità di crociera intorno ai 4 nodi, per una barca a motore si deve considerare una velocità 4 o 5 volte superiore. In pratica ci si aspetta che un'imbarcazione a vela utilizzi indifferentemente entrambi i parametri, mentre un'imbarcazione a motore utilizzerà prevalentemente il parametro metrico. Il dispositivo passa automaticamente a questo funzionamento dal funzionamento Libero se è abilitata la funzione di tracking e l'imbarcazione si sta muovendo con una velocità superiore a 1 nodo (soglia ritenuta sufficiente in quanto il dispositivo è indirizzato principalmente ad imbarcazioni a vela), per poi tornare al funzionamento Libero quando una di queste due condizioni viene meno.

3.2 Funzioni ausiliari

Quanto appena esposto sui differenti funzionamenti del dispositivo è quanto traspare ad un qualsiasi utilizzatore del dispositivo. Ma questi da soli non sono in grado di garantire il corretto svolgimento di tutte le funzionalità del dispositivo. Vi sono infatti alcune routine che il cliente non vede e non può gestire, le quali garantiscono che il sistema lavori al meglio.
Di seguito vengono riportate tali funzioni:

- Routine di gestione delle richieste: La gestione delle comunicazioni GSM/GPRS è strutturata come uno stack FIFO circolare nel quale vengono inserite le richieste sviluppate durante i vari stati di funzionamento e dal quale si possono prelevare le stesse in modo asincrono con l'esecuzione del funzionamento principale, così da non incorrere in problemi di temporizzazione tra le varie funzioni.

- Routine del modulo GSM/GPRS/GPS: Una volta inserite le richieste nello stack, questa routine ha il compito di gestirle una ad una. Tutte le comunicazioni sui canali GSM/GPRS/GPS vengono gestite da questa routine, evitando così conflitti tra le varie funzioni.

- Routine di debug: Questa routine gestisce alcuni LED utili al fine di ottenere un sistema di debug minimale interpretabile anche dal cliente, così da ottenere delle informazioni sul funzionamento del sistema.

- Routine di memorizzazione: Il dispositivo è in grado di memorizzare in una memoria FLASH lo stato dell'imbarcazione (con data e ora, il tipo di funzionamento, tutti gli ingressi e le uscite, la posizione). Questa routine garantisce al sistema le funzioni di scrittura, lettura e cancellazione della memoria.

Dalle funzioni si sono escluse le comunicazioni I^2C e RS232, in quanto non ritenute utili ai fini del funzionamento. I motivi di questa scelta sono legati alla scarsa applicabilità di un sistema di debug sul campo, in quanto il dispositivo viene solitamente montato in luoghi difficilmente raggiungibili così che i malintenzionati non riescano ad individuarlo. Un altro motivo è legato al fatto che si è optato per un sistema di debug basato sul server. Ovvero qualsiasi modifica o evento rilevato dal dispositivo, vengono registrati in un database consultabile da Aitech sul server. Questo permette di avere un servizio di assistenza al cliente molto più rapido ed economico, in quanto non risulta necessario mandare un tecnico sull'imbarcazione del cliente per rilevare il guasto.
Nel caso questo metodo di debug non dovesse risultare sufficiente, si può comunque collegare il dispositivo ad un PC attraverso il sistema di programmazione, questo però comporta la necessità di rimuovere il dispositivo dall'imbarcazione e il disassemblaggio da parte di un tecnico autorizzato da Aitech.

3.3 Sistemi di comunicazione

Il vecchio modello comunicava con l'utente attraverso una serie di codici inviati via SMS. Per il nuovo dispositivo si è mantenuto tale metodo di comunicazione, seppur ridefinendo tali codici, ed è stato aggiunto un metodo di comunicazione basato sulla rete GPRS.
Per renderlo più robusto e sicuro, si è progettato il sistema in modo tale che non venga instaurata una comunicazione con qualsiasi numero di telefono che provi a mettersi in contatto con il dispositivo, ma vi sono solo alcuni numeri autorizzati alla comunicazione con lo stesso. Infatti è possibile configurare un massimo di 5 *numeri abilitati* e altri 5 *numeri allertabili*. Come si deduce dai nomi, i primi sono autorizzati a inviare messaggi al dispositivo, il quale li elaborerà correttamente eseguendo le azioni descritte dai codici in essi contenuti. Gli altri 5 numeri sono

quelli ai quali il dispositivo segnalerà via SMS il verificarsi di eventi anomali. Solitamente i due gruppi di 5 numeri coincidono, ma l'utente non è obbligato a rispettare tale norma ed è libero di inserire numeri differenti.
Questo sistema di numeri abilitati risponde alle richieste di sicurezza, infatti l'applicazione reagisce solo con i numeri memorizzati. Se arrivano messaggi (anche contenenti testi riconosciuti dal dispositivo) da numeri non abilitati, vengono scartati, proseguendo con il normale funzionamento.

Per la comunicazione GPRS si è scelto di appoggiarsi ad un server esterno, principalmente perché gli operatori telefonici italiani non permettono alle proprie SIM di utilizzare indirizzi IP statici. Questo risulta essere un problema perché non permette di collegare in modo diretto un qualsiasi smartphone al dispositivo.
L'utilizzo di un server risolve questo problema, infatti i server hanno un IP statico per mezzo del quale può essere contattato in modo diretto.
La comunicazione è stata schematizzata in Figura 3.

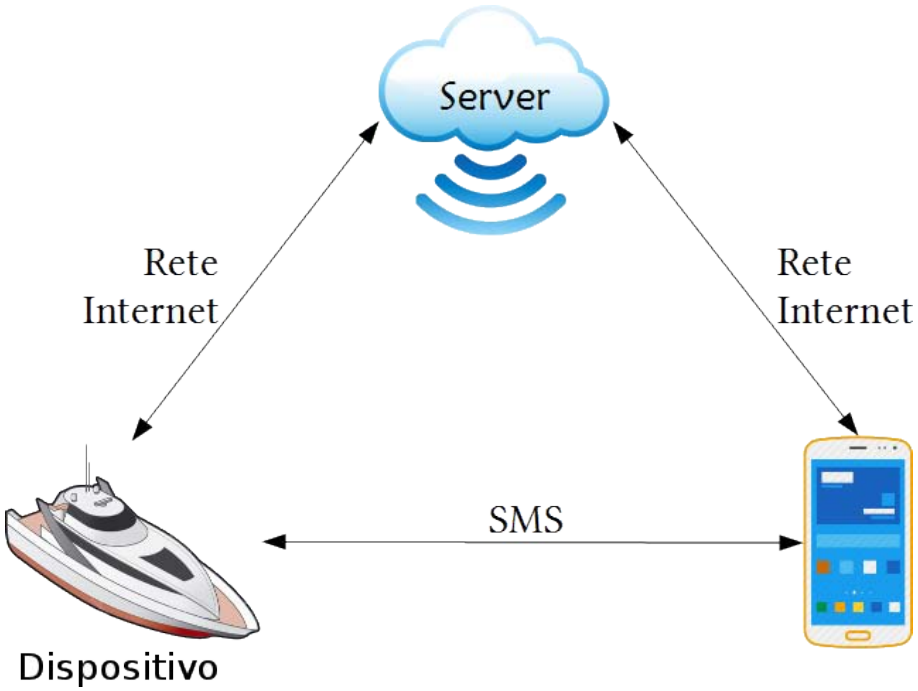


Figura 3: Schema connessione tra smartphone ed eKEEPER2

Questa soluzione rende il dispositivo un client del server. In base all'indirizzo che il dispositivo visita e ai parametri che invia, riceverà sotto forma di pagina internet le informazioni che il cliente ha trasmesso con il suo smartphone.
Per maggiori informazioni riguardo il funzionamento del server e della comunicazione internet, si faccia riferimento a [9].

3.3.1 Comunicazione GSM

Come già detto, il sistema prevede una comunicazione tramite SMS per permettere al cliente di monitorare il funzionamento e controllare il dispositivo da remoto.
Al fine di permettere al cliente di gestire le funzionalità del dispositivo, si sono previsti i seguenti codici:

- **COD01:x:y** questo codice viene utilizzato per reimpostare user name (x) e password (y) per l'accesso al server da parte del dispositivo
- **COD02:num1:num2:num3:num4:num5** questo codice viene utilizzato per reimpostare i *numeri abilitati*
- **COD03:num1:num2:num3:num4:num5** questo codice viene utilizzato per reimpostare i *numeri allertabili*
- **COD04:** non configurato
- **COD05:x:y** questo codice viene utilizzato per il telecontrollo delle tre utenze collegate al dispositivo. Il valore x (compreso nell'intervallo [1,3]) indica l'utenza che si vuole comandare, mentre il valore y attiva (y=1) o disattiva (y=0) l'uscita scelta.
- **COD06:x** questo codice abilita (x=1) o disabilita (x=0) il funzionamento *Manutenzione*
- **COD07:x** questo codice abilita (x=1) o disabilita (x=0) la possibilità di entrare in modalità di tracking del viaggio
- **COD08:x** questo codice abilita (x=1) o disabilita (x=0) il funzionamento *Protetto*
- **COD09:x** questo codice richiede al dispositivo l'ultimo stato dell'imbarcazione memorizzato e nel caso di x=1 anche l'ultima posizione registrata. La risposta viene inviata via SMS
- **COD10:x** questo codice richiede al dispositivo i *numeri abilitati* (x=0) o i *numeri allertabili* (x=1) memorizzati, il quale invierà la risposta via SMS

Ogniqualvolta il dispositivo riceve un SMS, prima di estrarre il contenuto del messaggio, verifica che il numero del mittente rientri tra i *numeri abilitati*. Solo nel momento in cui tale verifica da esito positivo, ne viene letto il contenuto, in caso negativo lo si trascura. Una volta estratto il contenuto del messaggio, si procede con un'ulteriore verifica sul testo. Se il codice rientra tra quelli elencati in precedenza, il dispositivo provvede con l'elaborare l'informazione ricevuta. In caso contrario, il dispositivo trascura l'SMS ricevuto.
Una volta ricevuta l'informazione corretta, il dispositivo eseguirà l'azione richiesta, subito seguita dalla generazione e invio del messaggio SMS di risposta.
Si sono quindi previste delle forme standard di risposta ad ogni codice inviato dal cliente, le quali sono riassunte in Tabella 2.

Codice Ricevuto	Risposta prodotta	Descrizione
COD01:x:y	COD01:User e Password Aggiornati	
COD02:num1$\cdots$	COD02:Numeri abilitati aggiornati	
COD03:num1$\cdots$	COD03:Numeri allertabili aggiornati	
COD04	/	
COD05:x:y	COD05:x:y	Risposta scritta leggendo stato vero
COD06:x	COD06:0:Disattivazione Manutenzione	Caso x=0
	COD06:1:Manutenzione Attiva	Caso x=1
COD07:x	COD07:0:Disattivazione Tracking	Caso x=0
	COD07:1:Attivazione Tracking	Caso x=1
COD08:x	COD08:0:Disattivazione Protezione	Caso x=0 e Funzionamento Protetto
	COD08:0:Disattivazione Allarme e Protezione	Caso x=0 e Funzionamento Allarme
	COD08:0:Protezione non Applicabile in MOV	Caso x=1 e Funzionamento MOV
	COD08:1:Protezione Attiva	Caso x=1 e Protezione abilitabile
COD09:x	COD09:*"stato imbarcazione"*	Caso x=0
	COD09:*"stato imbarcazione""posizione"*	Caso x=1
COD10:x	COD10:num1:num2:num3:num4:num5	Caso x=0: num1$\cdots$5 *numeri abilitati*
	COD10:num1:num2:num3:num4:num5	Caso x=1: num1$\cdots$5 *numeri allertabili*

Tabella 2: Risposte standard per comunicazione SMS

3.3.2 Comunicazione GPRS e interfaccia utente

Un punto di forza del dispositivo, vuole essere la comunicazione con lo stesso e l'interfaccia utente, che non richieda di ricordare il significato di ogni singolo codice, ma che ne semplifichi l'utilizzo, valendosi anche di un ambiente grafico.

Come già detto, la comunicazione GPRS tra il dispositivo e lo smartphone del cliente avviene attraverso un server.
Nel dispositivo sono già salvati gli indirizzi internet da contattare per la comunicazione, che saranno uguali per ogni ekeeper costruito. Tuttavia, ogni dispositivo ha dei parametri differenti che vengono inviati quando si stabilisce la comunicazione con il server. Questo permette al server stesso di distinguere le informazioni che riceve ed invia, potendo gestire tutti i dispositivi, senza incorrere in errori.
In questa tesi non verranno approfonditi gli indirizzi e i parametri trasmessi per motivi di riservatezza aziendale.

La comunicazione con il server avviene a sua volta con una serie di codici che vengono interpretati dagli interlocutori. Questi risultano più complessi ed elaborati rispetto a quelli presentati per la comunicazione GSM. Di conseguenza è necessario, in particolar modo per questo tipo di comunicazione, predisporre un'interfaccia adeguata per il cliente.
A tal fine è stata sviluppata un'applicazione *Android* che rende più agevoli le comunicazioni con il dispositivo.
Vista la flessibilità delle applicazioni per smartphone, è stato possibile implementare sia la comunicazione GSM che la comunicazione GPRS. Si è quindi creata un'interfaccia per l'utente che che permette di eseguire con semplicità tutte le azioni esposte in precedenza per gli SMS e molte altre. L'applicazione è inoltre in grado di determinare se il dispositivo sia raggiungibile attraverso la comunicazione GPRS, in modo da scegliere la comunicazione meno dispendiosa.

L'interfaccia dell'applicazione che si presenterà al cliente è composta principalmente da quattro pagine, che verranno qui brevemente presentate:

Scermata Home

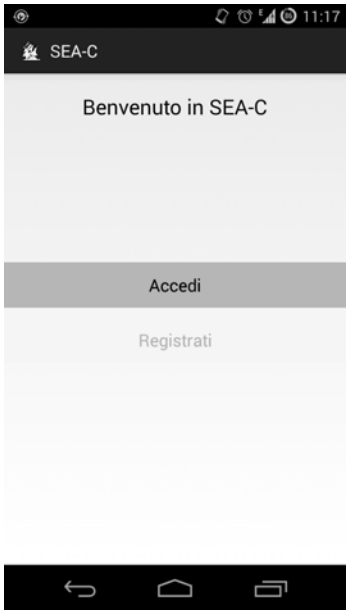
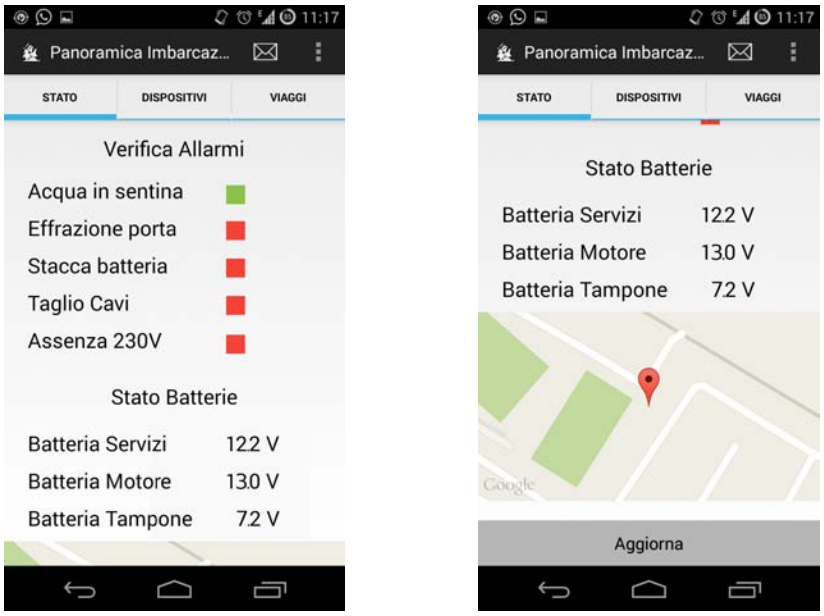


Figura 4: Schermata *Home* e di *Login* all'applicazione

In Figura 4 è riportata la Schermata *Home*, ovvero la prima pagina che si presenta una volta avviata l'applicazione sullo smartphone.
Da questa schermata è possibile effettuare l'accesso alle pagine successive, protette da username e password.

Schermata di monitoraggio dell'imbarcazione



(a) Parte superiore (b) Parte inferiore

Figura 5: Schermata di monitoraggio dell'imbarcazione

In Figura 5 sono riportate la parte superiore ed inferiore della Schermata di monitoraggio dell'imbarcazione.
Nella parte più alta sono riportati lo stato dei vari eventi anomali che il dispositivo sta monitorando. Si sono utilizzati i colori verde in caso non sia rilevato tale evento e il rosso in caso l'evento corrispondente sia stato rilevato. In questa sezione si possono ottenere informazioni sugli eventi/ingressi del dispositivo, qualunque sia lo stato di funzionamento impostato.
Oltre alla rilevazione degli eventi anomali, si ottengono informazioni riguardo le tensioni delle batterie che vengono monitorate dal dispositivo.
Infine viene visualizzata sulla mappa, l'ultima posizione rilevata.

Scermata di gestione da remoto

In Figura 6 è riportata la Schermata di gestione da remoto dei dispositivi collegati alle tre uscite.
Attraverso questa pagina, il cliente può controllare i dispositivi con un semplice tocco dello schermo, senza dover più ricordare il codice SMS. Se il dispositivo è spento, il pulsante risulta di colore blu, mentre una volta acceso, il pulsante risulterà di colore verde.
Vengono riportate inoltre informazioni sulla temperatura ed sull'umidità rilevate da ekeeper2, con il relativo sensore, fornendo anche la possibilità di visualizzare uno storico dell'andamento delle stesse. Infine vi sono due pulsanti, che con il semplice tocco del primo pone il sistema in funzionamento Protetto, oppure abilita la possibilità di entrare in funzione di tracking, risparmiando nuovamente al cliente la necessità di ricordare i codici SMS corrispondenti.

Figura 6: Schermata di gestione da remoto dei dispositivi

Schermata di visualizzazione dei viaggi

Figura 7: Schermata di visualizzazione dei viaggi

In Figura 7 è riportata la pagina di visualizzazione dei viaggi memorizzati dal dispositivo e caricati sul server. Questa funzione è disponibile solamente tramite la connettività GPRS, infatti risulterebbe troppo dispendioso inviare tutte le posizioni attraverso i classici SMS.
In questa pagina è possibile scegliere il viaggio in base alla data per poi visualizzarlo immediatamente sulla mappa sottostante.

La descrizione completa dello sviluppo dell'applicazione per smartphone Android è riportata in [9].

4 Progettazione prototipo

La fase della progettazione che viene qui descritta, riguarda la costruzione di un prototipo fisico sul quale sviluppare poi il software ed effettuare i primi test.

Quanto verrà presentato di seguito è la versione corretta del primo prototipo preparato. Tale risultato è frutto di numerose attività in grado di rispondere alle esigenze del progetto. Per la scelta di ognuno dei componenti si è effettuato un approfondito studio delle schede tecniche, considerando gli aspetti di soluzione, economici e di approvvigionamento.

4.1 Arduino

Per lo sviluppo del prototipo si è scelto di utilizzare un microcontrollore della gamma *Arduino*, come richiesto da Aitech.
Arduino offre una vasta gamma di microcontrollori. Dalle schede più semplici che montano processori ad 8 bit con ridotta capacità di calcolo, passando per processori con una maggior potenza fino a processori ARM a 32 bit. Tutti questi in varie dimensioni ed interfacce hardware.

Tra i modelli disponibili, si è optato per il microcontrollore *Arduino DUE* per i seguenti motivi:

- Numero di ingressi/uscite: Il sistema richiede un considerevole numero di ingressi/uscite per la gestione dei trasduttori in ingresso, degli attuatori in uscita e degli elementi che utilizzano protocolli di comunicazione quali I^2C, SPI, Comunicazione Seriale. Per questo motivo sono state escluse soluzioni più semplici ed economiche, quali ad esempio il modello *Arduino UNO*.
- Dimensione Flash Memory: In fase di programmazione si è riscontrato un problema relativo alla dimensione del codice, il quale ha superato i 32 KB: e questo ha richiesto un dispositivo in grado di gestire memorie di queste dimensioni.
- Dimensione SRAM: In fase di test si sono riscontrati dei problemi sulla gestione delle variabili, che si sono attribuiti alla necessità di una SRAM superiore agli 8 KB.

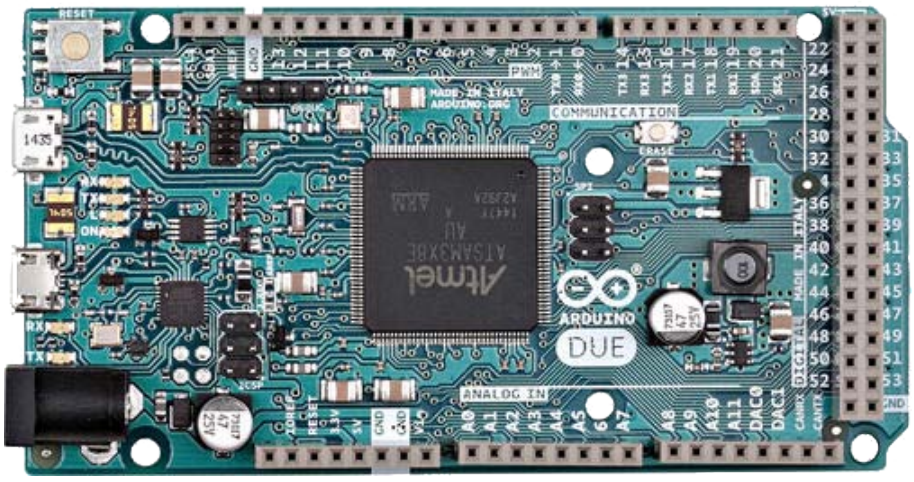


Figura 8: Arduino DUE

Arduino DUE si è rivelato la soluzione ideale, infatti è una scheda microcontrollore che si basa sul processore *Atmel SAM3X8E ARM Cortex-M3*, il quale ha una capacità di calcolo

adatta all'applicazione e ne garantisce ottime prestazioni.
Le caratteristiche salienti del processore che sono d'interesse per il progetto sono le seguenti:

- Tensione operativa: 3.3 V
- Tensione di ingresso: $7 - 12$ V
- Pin I/O digitali: 54
- Pin input anagoligici: 12
- Memoria Flash: 512 KB
- SRAM: 96 KB
- Frequenza di clock: 84 MHz

Per una più approfondita descrizione, si rimanda alla pagina web di riferimento di Arduino DUE [1].

4.2 Shield GSM/GPRS/GPS

Per la comunicazione da remoto si è ricorsi ad uno shield per *Arduino*, pre-assemblato, basato sul modulo *SIM908* prodotto dalla *SIMCom*, riportato in Figura 9. Questo è un modulo Quad-Band GSM/GPRS che combina anche la tecnologia GPS per la navigazione satellitare. Per maggiori informazioni si faccia riferimento alla pagina web del prodotto [2].

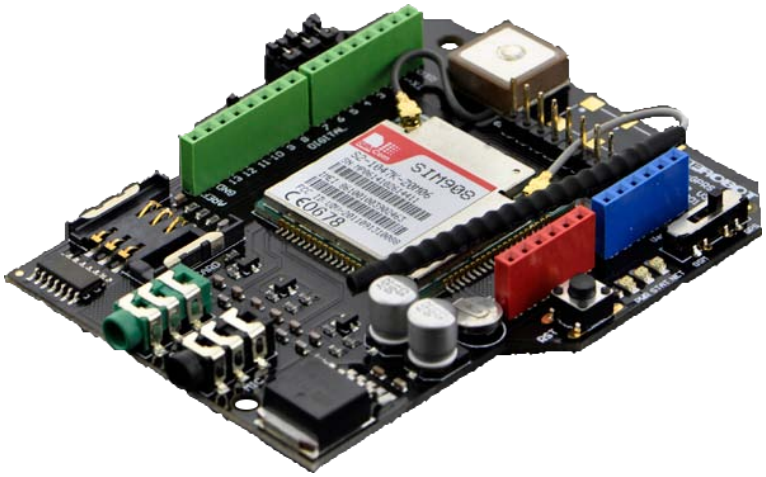


Figura 9: Scheda di comunicazione GSM/GPRS/GPS basata su modulo *SIM908*

Questo modulo comunica con Arduino attraverso la comunicazione Seriale. Lo shield riunisce il modulo *SIM908*, un regolatore di tensione per la corretta alimentazione del modulo, un supporto per l'inserimento della scheda SIM dell'operatore, le antenne e un buffer per adattare le tensioni sulla comunicazione Seriale. Questa scheda è nata come modulo aggiuntivo per*Arduino*, al fine di incrementarne le funzioni e renderlo simile ad un telefono cellulare programmabile.
L'interfaccia adatta per la comunicazione con *Arduino* e il funzionamento come telefono cellulare unito alla geo-localizzazione rendono questa scheda una buona soluzione per l'applicazione oggetto di questa tesi.

4.3 Interfaccia dispositivi a bordo

Nonostante *Arduino* preveda un'interfaccia tra il processore e il mondo esterno, questo non è sufficiente a soddisfare le richieste di progetto.
Il dispositivo necessita di un'interfaccia che permetta al cliente di collegare i vari apparecchi in modo corretto, senza rischiare di confondere due pin troppo vicini tra loro.
Inoltre, dalle caratteristiche tecniche di *Arduino DUE*, risulta che la tensione di lavoro del processore è di soli 3.3V, non adatta per la gestione di apparecchi esterni, sia in ingresso che in uscita.

Per questi motivi si è provveduto a realizzare su millefori una scheda che riunisse l'interfaccia tra Arduino e i trasduttori/attuatori, la gestione dell'alimentazione e una memoria FLASH per la registrazione dei viaggi.
La scheda, assemblata a mano, è riportata in Figura 10.

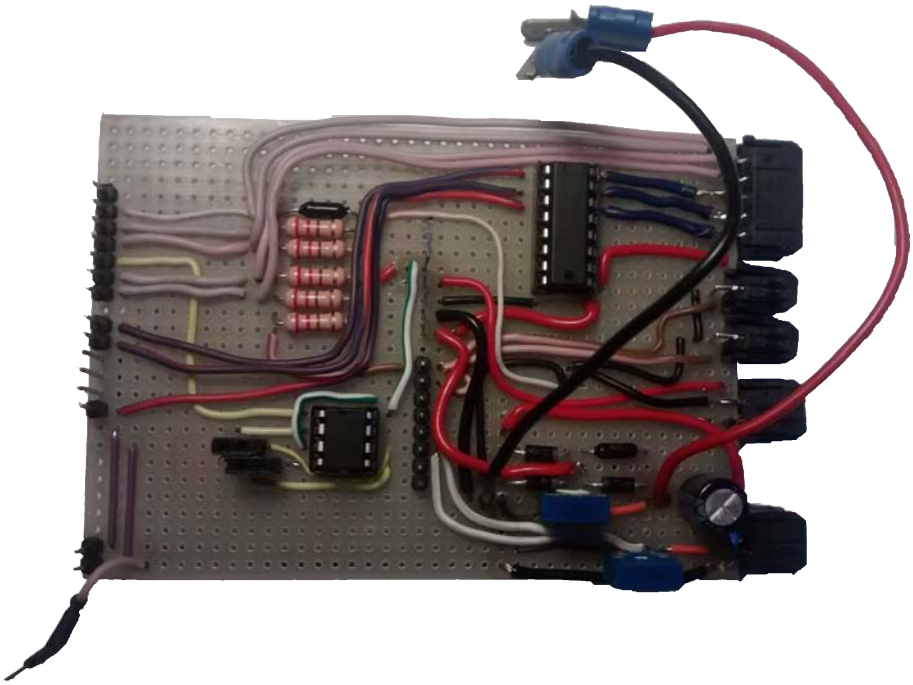


Figura 10: Interfaccia per dispositivi a bordo

Vengono quindi analizzati i principali componenti di questa scheda.

4.3.1 Ingressi

I trasduttori sono elementi esterni al prototipo, installati nell'imbarcazione su richiesta del cliente.
Per i quattro ingressi previsti, i rispettivi trasduttori si possono modellizzare come interruttori monostabili, normalmente chiusi nel caso degli ingressi "Ausiliario", "Apertura Porta" e "Attivazione Staccabatteria", mentre risulta normalmente aperto nel caso dell'ingresso "Acqua in Sentina".

Lo schema applicato per gli ingressi è quello riportato in Figura 11.
Si è scelto di utilizzare questa configurazione per tutti gli ingressi, a prescindere dal funzionamento normalmente aperto o normalmente chiuso, e gestire i due casi via software.

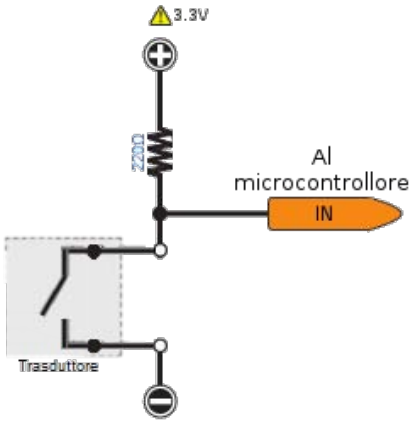


Figura 11: Schema ingressi

4.3.2 Lettura delle tensioni delle batterie

Nel sistema è stata prevista una batteria tampone, che deve subentrare per garantire la routine d'emergenza nel caso dovesse venir meno l'alimentazione prelevata dalla batteria servizi dell'imbarcazione.
Lavorando con delle batterie, il sistema implementa un'utile funzione di monitoraggio delle tensioni.
Le tensioni che si vogliono misurare variano nel tempo abbastanza lentamente da poter considerarle costanti. Si è scelto quindi di utilizzare un partitore di tensione con rapporto di partizione noto, al fine di rendere la tensione delle batterie leggibile con una porta analogica di *Arduino*.
In Figura 12 è riportato lo schema di collegamento per la lettura di un valore analogico.

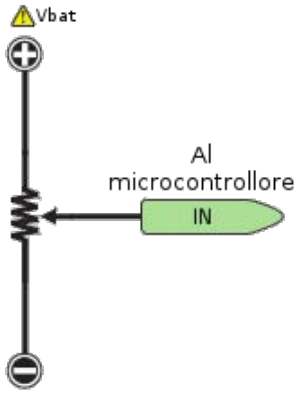


Figura 12: Schema lettura tensione batterie

4.3.3 Uscite

Gli attuatori sono elementi esterni al prototipo, installati nell'imbarcazione su richiesta del cliente. Si raccomanda comunque l'utilizzo di relè con bobina a 12 V, corrente massima 500 mA e contatto a 230 V o differente, in base alle necessità del cliente.
Tuttavia il microcontrollore utilizzato ha una potenza e una tensione non sufficienti a comandare direttamente gli attuatori.
Per ovviare a tale problema si ricorre al componente elettronico ULN2003. Questo componente è un array di transistor ad effetto Darlington ad elevate tensioni e correnti. Grazie a tale

componente si è in grado di controllare attuatori fino a 50 V e 500 mA.

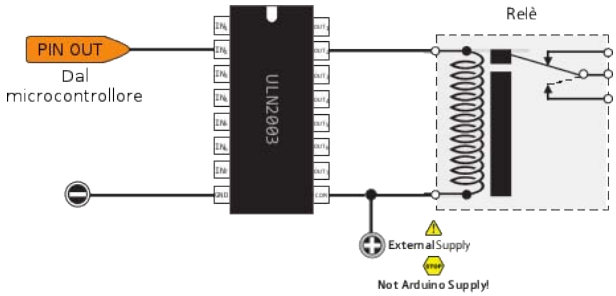


Figura 13: Schema uscite con interfaccia ULN2003

Lo schema applicato per le uscite è quello riportato in Figura 13.
Si è scelto di utilizzare questa configurazione per tutte le uscite. È importante osservare che l'alimentazione collegata all'integrato ULN2003 e al relè è esterna ad *Arduino* proprio per l'energia che potrebbe richiedere tale applicazione.
Nel dispositivo esposto in questo trattato, tale alimentazione è prelevata direttamente dal connettore di alimentazione del dispositivo stesso, senza conversioni.
Si ha però ai capi del relè una tensione che varia in base alla tensione con cui si alimenta il dispositivo.
È compito del cliente predisporre dei relè con bobina adeguata alla tensione con cui si alimenta il dispositivo.

4.3.4 Memoria FLASH

Il sistema ha necessità di memorizzare delle variabili in modo non volatile, quali i numeri di telefono da contattare, o determinati dati di configurazione. Inoltre si necessita di memoria dove registrare le coordinate geografiche che si percorrono durante i viaggi e mantenerle fino al momento in cui, una volta terminati, verranno caricati sul server.
Si è scelto di utilizzare una memoria FLASH da 8 MB, suddivisa in blocchi da 4 kB, grazie alla semplicità con cui si svolgono le funzioni di scrittura/lettura/cancellazione e alla possibilità di utilizzare la comunicazione SPI.
In Figura 14 è riportato lo schema di collegamento utilizzato per la memoria.

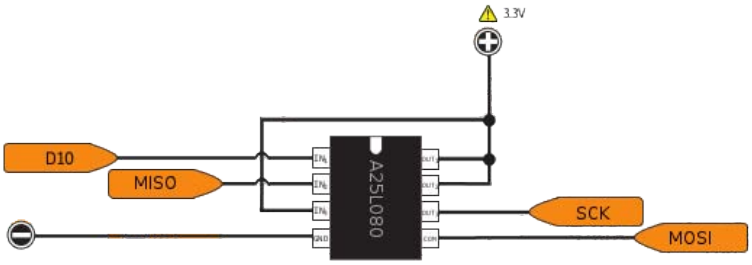


Figura 14: Schema memoria FLASH con comunicazione SPI

5 Programmazione prototipo

Per la programmazione del processore di *Arduino DUE* si può utilizzare l'ambiente di sviluppo messo a disposizione dal produttore del microcontrollore. Il linguaggio di programmazione si basa sul linguaggio C++, ma è stato adattato alle schede in questione. Per maggiori informazioni riguardo il linguaggio di programmazione si faccia riferimento alla pagina web [4].

5.1 Analisi e implementazione della struttura principale

Di seguito viene presentato come è stato strutturato il programma del microcontrollore. In particolare viene approfondita la struttura principale che è stata implementata, ovvero la gestione dei differenti tipi di funzionamento descritti in Sezione 3.1.

5.1.1 Automa a Stati Finiti Deterministico

Il funzionamento che viene richiesto al dispositivo risulta essere ripetitivo, con un insieme limitato di eventi da gestire e di azioni da effettuare.
In base a queste osservazioni risulta quindi adeguato far ricorso agli *Automi a Stati Finiti Deterministici*.
Questo automa è un modello matematico che permette di descrivere con precisione e in maniera formale il comportamento di molti sistemi. È concepito come una macchina astratta che può essere composta da uno o più stati (in numero finito). Tale macchina può essere in un solo stato alla volta e può passare da uno stato ad un altro in seguito ad un evento o una condizione, chiamata transizione [5].

La gestione dei tipi di funzionamento, ovvero la struttura principale del dispositivo, la si può rappresentare quindi con il seguente *automa a stati finiti deterministico*

$$\mathcal{G} = (X, E, f, \Gamma, x_0, X_m)$$

Gli elementi che compongono l'automa sono riportati di seguito:

- X è l'insieme di tutti gli stati. X = {S1, S2, S3, S4, S5, S6} dove S1=Libero, S2=Protetto, S3=Allarme, S4=Allarme Zona, S5=Manutenzione, S6=Viaggio. I singoli stati sono in realtà sub-automi a stati finiti deterministici a loro volta. Questi non verranno approfonditi in questo trattato per motivi aziendali.
- E è l'insieme degli eventi possibili (qui per semplicità di notazione si sono indicati con lo stesso nome "SMS/App" vari eventi, ma durante l'esecuzione, questi sono distinti dall'informazione contenuta nel pacchetto). E = {a.1, a.2, a.3, a.4, a.5, a.6, a.7, a.8, b, c, d.1, d.2, d.3, d.4, e, f}, dove: a.i="SMS/App" con i che dipende dall'informazione del pacchetto , b="TrackON+vel>3nodi", c="TrackOFF/vel<3nodi", d.l="Evento anomalo" con l che dipende dall'evento, e="Fuori Zona", f="1h".
- f è la funzione di transizione.
- Γ rappresenta tutte e sole le funzioni di transizione attive per ogni stato. Γ e f vengono meglio rappresentati in Tabella 3.
- x_0 =Libero è lo stato iniziale.
- $X_m = \emptyset$ è l'insieme degli stati finali. Nel caso in esame, tale insieme risulta essere vuoto in quanto il sistema non ha un obbiettivo da raggiungere, ma dovrà continuare a lavorare in modo ciclico per un tempo idealmente infinito.

f	a				b	c	d				e	f
	a.1	a.2	a.3	a.4			d.1	d.2	d.3	d.4		
S1	S2	/	S5	/	S6	/	/	/	/	/	/	/
S2	/	S1	S5	/	/	/	S3	S3	S3	S3	S4	/
S3	/	S1	S5	/	/	/	/	/	/	/	S4	S2
S4	/	S1	S5	/	/	/	/	/	/	/	/	S2
S5	S2	/	/	S1	/	/	/	/	/	/	/	/
S6	/	/	S5	/	/	S1	/	/	/	/	/	/

Tabella 3: Funzioni di transizione

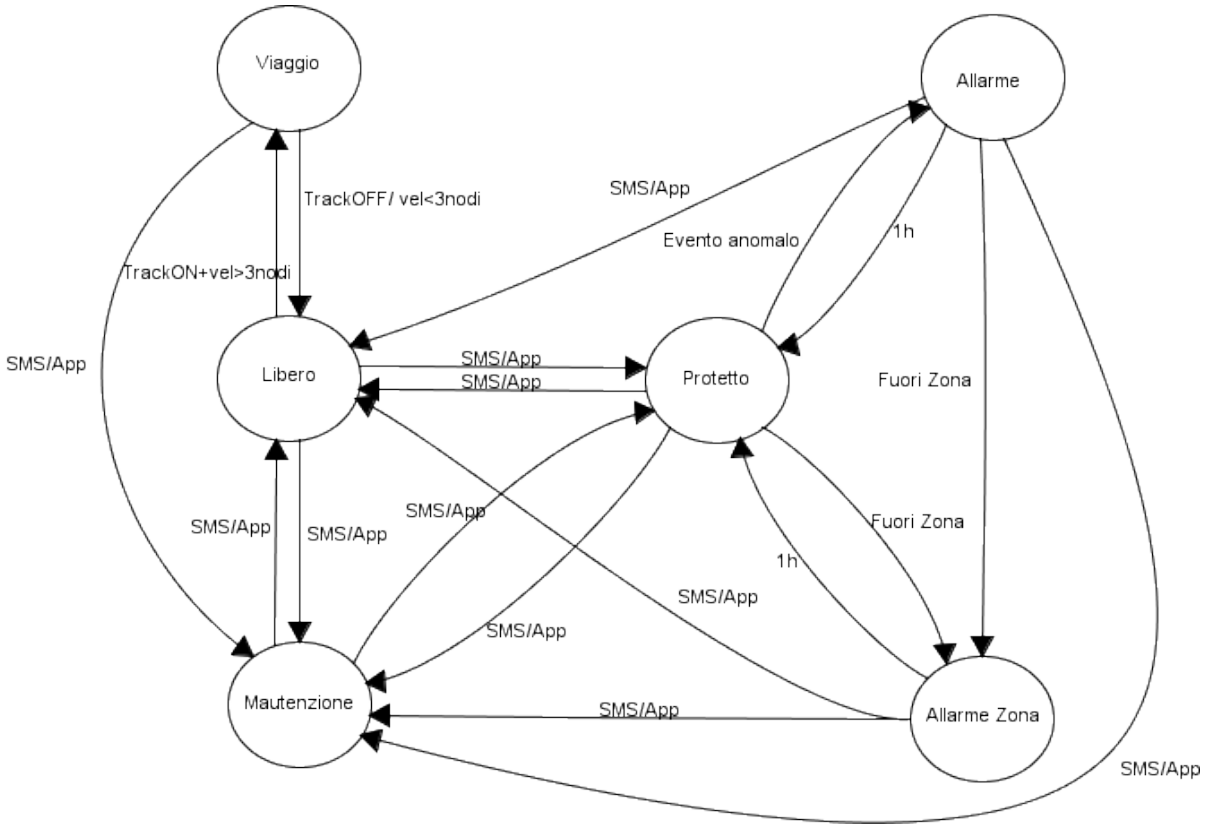


Figura 15: Automa a Stati Finiti Deterministico

In Tabella 3 e in Figura 15 si può riconoscere la classica rappresentazione per l'*automa a stati finiti deterministico* sopra descritto. Oltre a descrivere in modo completo il funzionamento del sistema, tale descrizione risulta essere un valido punto di partenza per l'implementazione, come verrà mostrato nella Sezione 5.1.2.

5.1.2 Implementazione dell'Automa

Per l'implementazione dell'automa presentato nella sezione precedente, si è costruita una struttura che ponesse attenzione alle seguenti caratteristiche:

- Utilizzare una struttura quanto più fedele all'automa, che si ripetesse in tutti gli stati, con gli adeguamenti del caso;
- Mantenere una struttura chiara, in modo da poter essere compresa anche dopo tempo;
- La necessità di non bloccare il ciclo del microcontrollore in attesa degli eventi, in quanto questo deve eseguire altre routine esterne all'automa.

Una struttura che risponda al meglio alle caratteristiche sopra descritte, considerando la rappresentazione grafica di automa generico, dovrà racchiudere lo stato e tutti gli archi uscenti dallo stesso. Questo si traduce nella seguente serie di compiti:

1. Eseguire le azioni associate allo stato corrente appena lo si raggiunge;
2. Verificare la presenza delle condizioni necessarie per passare ad un altro stato;
3. Effettuare il passaggio allo stato corrispondente.

Particolare attenzione si è posta al fatto che non si vuole bloccare il ciclo del microcontrollore in attesa di un evento per passare da uno stato all'altro. Quindi si è previsto che il sistema possa uscire dalla struttura che si sta descrivendo, per ritornare al ciclo successivo al punto dell'esecuzione in cui era arrivato in precedenza. Questo rischia di creare difficoltà al sistema nel riconoscere se è il primo accesso che fa allo stato in esame, o se vi è rientrato dopo esservi uscito per non bloccare il processo. Quindi per non perdere l'informazione se si è appena giunti allo stato corrente e si deve eseguire l'azione o se l'azione è già stata eseguita, si utilizza una variabile booleana chiamata "abilitato" che viene posta a "true" prima di passare da uno stato al vicino per poi essere posta a "false" non appena l'azione associata allo stato è stata eseguita. Dettagli, a questo riguardo, sono riportati in Listing 1.

Listing 1: Implementazione della struttura che riproduce uno stato dell'automa

```
// Implementazione dello stato
if (stato == 1){

    // Accesso allo stato
    if (abilitato){
        // Azioni da eseguire
        azione();
        // Azioni eseguite
        abilitato = false;
    }

    // Verifica condizioni
    if (condizione){
        // Passaggio allo stato vicino
        stato = 2;
        // Abilitazione delle azioni dello stato vicino
        abilitazione = true;
    }
}
```

Finché la variabile "stato" rimane uguale a 1, il programma esegue quanto contenuto nella struttura "if (stato==1)", ma solo la prima volta che tale condizione è verificata, verrà eseguito il contenuto della struttura "if (abilitato)". Quest'ultima struttura contiene tutte le azioni associate allo stato corrente e compare una volta per stato. La struttura "if (condizione)" controlla il verificarsi dell'evento "condizione" e gestisce il passaggio allo stato riassegnando il valore corrispondente alla variabile "stato". Questa struttura è presente in numero pari al numero di transizioni attive associate allo stato corrente.

Questa implementazione risulta semplice in quanto rispecchia chiaramente quanto rappresentato dall'automa. Un altro pregio è la semplicità con cui si possono applicare le modifiche. Per aggiungere funzionalità è sufficiente aggiungere uno stato simile a quello qui riportato e aggiungere le strutture "if (condizione)" appropriate negli stati vicini dai quali si può giungere al nuovo. Si presta quindi per aggiungere funzionalità per futuri sviluppi dell'applicazione.

5.1.3 Tipo di ciclo

Il dispositivo è un sistema di allarme real-time.
Tuttavia i tempi caratteristici dei sistemi di comunicazione sui quali si basa, sono dell'ordine delle decine di secondi.
Non risulta quindi produttivo spingere il dispositivo a lavorare a tempi molto stringenti.
Come verrà mostrato in Sezione 6.1.1, il tempo di ciclo medio del sistema si assesta sui $2 \div 3$ s.
La differenza tra il tempo di ciclo e i tempi per la comunicazione risulta di un ordine di grandezza.
Sulla base di queste considerazioni, si è giunti alla conclusione che il tipo di ciclo più si adatta all'applicazione qui presentata è quello *Sincrono d'Ingresso - Sincrono d'Uscita*.
Un sistema basato su un tipo di ciclo *Sincrono d'Ingresso* legge tutti gli ingressi all'inizio del ciclo e memorizza le informazioni in relative immagini. All'interno del programma, quando viene richiesto il valore di un ingresso non verranno interrogati gli ingressi fisici, ma le relative immagini aggiornate all'inizio del ciclo. Il ritardo massimo tra il cambio dello stato di un ingresso e la rilevazione da parte del sistema è pari ad un tempo di ciclo.
Allo stesso modo, durante l'esecuzione del programma in un sistema basato su un tipo di ciclo *Sincrono d'Uscita*, i valori di controllo per le uscite vengono scritti in un'immagine. Solo al termine del ciclo i valori memorizzati in queste immagini verranno effettivamente applicati alle uscite fisiche.

Per quanto riguarda le comunicazioni GSM/GPRS/GPS, il sistema si comporta in modo differente.
Si considerino tutte le informazioni che si possono ricevere dal modulo *SIM908* come ingressi e tutte le informazioni che si forniscono al modulo *SIM908* per inviarle, in forma di SMS o di pacchetto su protocollo internet, come uscite.
Il tipo di ciclo risulta essere *Sincrono d'Ingresso - Asincrono d'Uscita*.
Gli ingressi "posizione GPS" e "SMS in ingresso" si possono considerare come "Ingressi Sincroni", infatti il sistema va ad interrogare ed elaborare questi all'inizio del ciclo, memorizzando le informazioni in relative immagini che vengono consultate durante l'esecuzione del programma.
L'ingresso "comunicazione GPRS in ingresso" risulta anch'esso sincrono. Infatti, è il dispositivo che stabilisce una connessione con il server, ottenendo così il contenuto della pagina web. Questo viene però elaborato solo all'inizio del ciclo, memorizzando le informazioni nelle relative immagini, come per tutti gli altri ingressi. Quindi l'informazione che si riceve dalla comunicazione GPRS in un ciclo, viene elaborata ed utilizzata come ingresso solo al ciclo successivo.
Le uscite invece, per questioni tecniche che verranno approfondite nella Sezione 5.2, non posso-

no essere implementate come uscite sincrone. Queste vengono inserite in una coda di esecuzione (trattata in Sezione 5.2.4) e il dispositivo le elabora in sequenza in modo asincrono all'intero sistema.
L'esecuzione di queste uscite può richiedere anche più cicli, motivo per cui si è impossibilitati nell'implementarle come uscite sincrone.

5.2 Comunicazione GSM/GPRS/GPS

La comunicazione GSM/GPRS/GPS è gestita dal modulo *SIM908*.
In Figura 16 è riportato il diagramma di funzionamento di tale modulo.
I componenti del diagramma d'interesse per il dispositivo presentato in questo scritto sono *UART*, *GPS Receiver* e *Radio Frequency*, i quali verranno presentati ed analizzati di seguito.

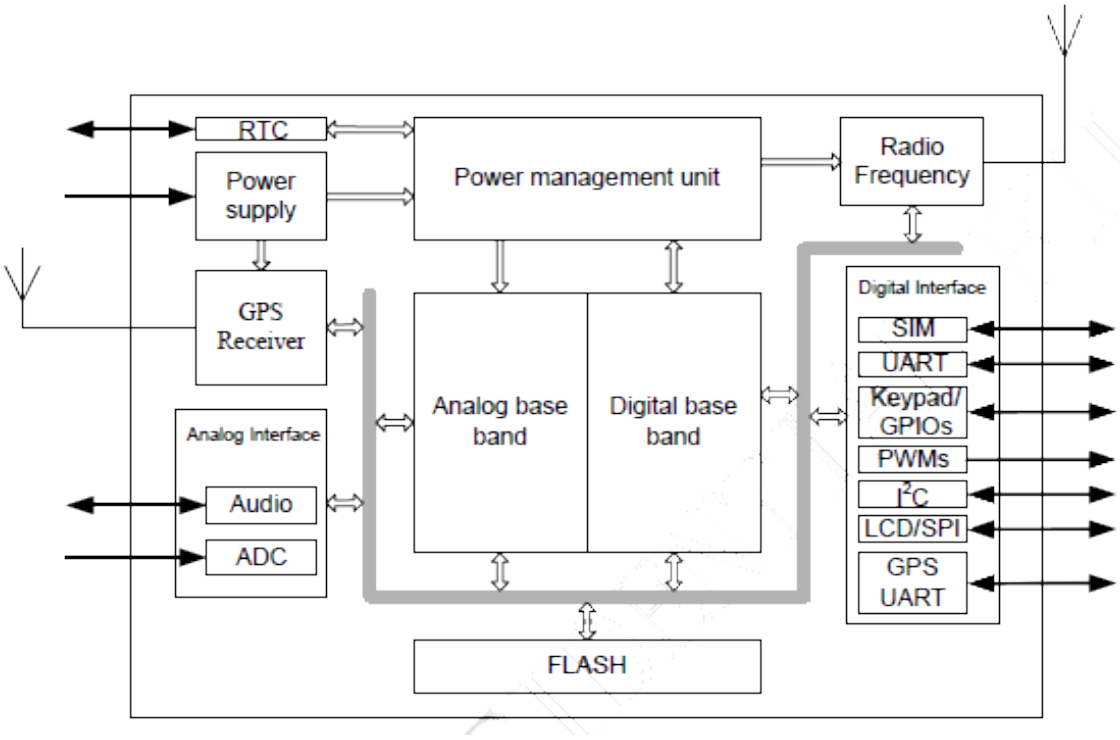


Figura 16: Diagramma Funzionale SIM908

5.2.1 Comunicazione con il modulo SIM908

Il modulo *SIM908* comunica con Arduino attraverso la comunicazione Seriale gestita dal dispositivo *UART* (Universal Asynchronous Receiver/Transmitter) implementato. *UART* è un dispositivo che converte i flussi di bit di dati da un formato parallelo a un formato seriale asincrono o viceversa [6]. Questo dispositivo è presente anche nel microcontrollore *Arduino* in corrispondenza dei pin riservati alla comunicazione Seriale.

Il sistema di comunicazione tra *Arduino* e il *SIM908* si basa su dei comandi detti *AT Commands* che vengono trasmessi attraverso la comunicazione seriale.
Per semplicità si può considerare il sistema al pari di un sistema *Master-Slave*. Infatti, *Arduino* trasmette il comando e il *SIM808* esegue l'azione associata al comando e risponde. Non avviene invece il contrario.
Il manuale degli *AT Commands* lo si può reperire sul sito del produttore [3].

Si riporta in Listing 2 un esempio di implementazione su *Arduino* del sistema di comunicazione.

Listing 2: Implementazione della comunicazione Seriale

```
// Invio del comando
Serial.println("AT"); // Gli AT Commands vengono inviati da Arduino come stringhe

// Ricezione della risposta
while(Serial.available()>0) { // Si verifica la presenza di comunicazione Seriale
    carattere = Serial.read(); // Si legge il carattere in arrivo
    risposta.concat(carattere); // Si compone la stringa ricevuta
}
```

5.2.2 Comunicazione GSM/GPRS

Il modulo *SIM908* implementa una comunicazione GSM/GPRS Quad-Band con tecnologia telefonica senza fili di seconda generazione (2G).
La tecnologia 2G risulta ormai datata (entrata in commercio nel 1991) e superata (si pensi che attualmente i dispositivi si appoggiano alla tecnologia di quarta generazione). Tuttavia l'applicazione qui descritta necessita di una comunicazione GPRS per l'invio/ricezione di pacchetti di dati mediamente contenenti alcune centinaia di caratteri d'informazione, e solo in casi particolari, l'invio di qualche migliaio di caratteri.
Si è analizzata la capacità di trasmissione dei dati della tecnologia 2G basato sul servizio GPRS (General Packet Radio Service). Questo è in grado di fornire una velocità di trasmissione pratica di 40 kbit/s, ovvero 5000 caratteri al secondo.
Sulla base della dimensione media dei pacchetti di dati trasmessi e sui tempi caratteristici del sistema, tale connessione si rivela un buon compromesso tra efficienza e costo.

La comunicazione tra *Arduino* e le funzionalità GSM e GPRS del modulo, avvengo tramite *AT Commands*, come per la comunicazione con il modulo stesso.
Le principali azioni che vengono compiute con l'ausilio del modulo *SIM908* sono:

- Effettuare una telefonata della durata di pochi squilli per la segnalazione urgente di un evento anomalo
- Ricevere ed inviare messaggi
- Stabilire una connessione GPRS con un server

Verranno di seguito presentati degli esempi per l'implementazione dal lato *Arduino* di queste funzioni.

Effettuare telefonate

Le azioni riguardanti le telefonate sono:

- Avvio di una telefonata
- Termine di una telefonata
- Verifica presenza telefonate in ingresso

Listing 3: Implementazione della funzione di chiamata

```
// Avvio telefonata
void avvioTelefonata() {

    // Composizione del comando AT
    String chiamata = "";
    chiamata.concat("ATD");
    chiamata.concat(numeroAllertabile);
    chiamata.concat(";");

    // Invio del comando al modulo con conseguente avvio della telefonata
    Serial.println(chiamata);

    return;
}
```

Nel progetto si è implementata la funzione per effettuare le telefonate (riportata in Listing 3), seppur non siano predisposti dispositivi per l'acquisizione e la riproduzione di segnali audio.
Il motivo della telefonata è legato ad un fattore psicologico, ovvero, una persona che riceve una telefonata è più probabile che controlli subito il telefono.
Viceversa, se arriva un messaggio, è probabile che si dia meno importanza quando si riceve la notifica sul telefono.

Sulla base di queste considerazioni, si è scelto di effettuare una telefonata di breve durata (quello che viene detto in gergo "squillo") prima di procedere con la segnalazione via SMS.
La durata della telefonata è gestita dall'implementazione fatta su *Arduino*

Listing 4: Implementazione della funzione per terminare le telefonate

```
// Termine telefonata
void termineTelefonata() {

    // Invio del comando al modulo con conseguente termine di tutte le telefonate in ingresso e in
        uscita
    Serial.println("ATH");

    return;
}
```

La funzione riportata in Listing 4 permette di terminare tutte le telefonate, in ingresso e in uscita.

Listing 5: Implementazione della funzione per verificare se vi sono telefonate in corso

```
// Verifica presenza telefonate in ingresso
boolean verificaTelefonata () {

   // Invio del comando per verificare la presenza di telefonate al modulo
   Serial.println("AT+CPAS");

   // Lettura della risposta
   String risposta = "";
   while(Serial.available()>0) { // Si verifica la presenza di comunicazione Seriale
   carattere = Serial.read(); // Si legge il carattere in arrivo
   risposta.concat(carattere); // Si compone la stringa ricevuta
   }

   // Elaborazione della stringa "risposta"
   int puntatorePiu = risposta.indexOf("+");
   // Estrazione carattere portante l'informazione desiderata
   risposta = risposta.substring(puntatorePiu + 7, puntatorePiu + 8);

   // Verifica chiamata in ingresso
   boolean chiamataIngresso = false;
   if (risposta.equals("3")) {
      chiamataIngresso = true;
   }

   return chiamataIngresso;
}
```

Nonostante non sia contemplato nel progetto, può capitare che il sistema venga contattato telefonicamente in modo involontario da parte del proprietario (o terzi). Il modulo notifica al microcontrollore la telefonata in ingresso trasmettendo via comunicazione Seriale la stringa "RING". Non è prevedibile l'istante in cui arriva la telefonata, quindi non lo è neppure il momento in cui compare la stringa. Bisogna quindi prevenire l'interferire di questa stringa con la comunicazione programmata.
Per rendere robusto il sistema si è ricorsi alla funzione verificaTelefonata , riportata in Listing 5. Eseguita prima di altre comunicazioni (in combinazione con la funzione termineTelefonata riportata in Listing 4), previene il comparire della suddetta stringa nella risposta del modulo.

Inviare/Ricevere SMS

Le azioni riguardanti gli SMS sono:

- Verifica presenza SMS da leggere
- Lettura SMS
- Invio SMS
- Cancellazione SMS

Listing 6: Implementazione della funzione per verificare il numero di messaggi in memoria da leggere

```
// Verifica presenza SMS da leggere
int verificaSMS() {
    // Invio del comando per verificare la presenza di SMS da leggere al modulo
    Serial.println("AT+CPMS?");

    // Lettura della risposta
    String risposta = "";
    while(Serial.available()>0) { // Si verifica la presenza di comunicazione Seriale
    carattere = Serial.read(); // Si legge il carattere in arrivo
    risposta.concat(carattere); // Si compone la stringa ricevuta
    }
    // Elaborazione della stringa "risposta"
    int puntatoreVirgola = risposta.indexOf(",");
    // Estrazione carattere portante l'informazione desiderata
    numeroSMS_Str = risposta.substring(puntatoreVirgola + 1,
        risposta.indexOf(",",puntatoreVirgola + 1));

    // Conversione da stringa a numero intero
    int numeroSMS = numeroSMS_Str.toInt();
    return numeroSMS;
}
```

Nella funzione verificaSMS riportata in Listing 6 si descrive un esempio di risposta che si ottiene al comando AT+CPMS? è +CPMS: "SM_P",0,20,"SM_P",0,20,"SM_P",0,20. Sulla base di questa informazione, è possibile fare un'elaborazione sulla stringa per estrarre il valore dei messaggi ricevuti, ancora da leggere, il quale è il valore riportato dopo la prima virgola a partire da sinistra.
Nel programma non interessa sapere quanti messaggi ci sono, ma sapere se ce ne sono. Quindi anche se viene estratto un solo valore, si riesce ad ottenere l'informazione voluta.

Listing 7: Implementazione della funzione per la lettura del contenuto dei messaggiricevuti

```
// Lettura SMS
String letturaSMS(int indiceSMS) {
    // Composizione del comando AT
    String comandoAT = "AT+CMGR=";
    comandoAT.concat(indiceSMS);

    // Invio del comando per leggere un SMS al modulo
    Serial.println(comandoAT);

    // Lettura della risposta
    String testoSMS = "";
    while(Serial.available()>0) { // Si verifica la presenza di comunicazione Seriale
    carattere = Serial.read(); // Si legge il carattere in arrivo
    testoSMS.concat(carattere); // Si compone la stringa ricevuta
    }
    return testoSMS;
}
```

Quando nell'esecuzione del programma si richiama la funzione letturaSMS, viene passato il valore indiceSMS che indica l'indice di posizione in memoria del modulo del messaggio che si vuole leggere. Tale parametro viene utilizzato per comporre il comando *AT* al quale verrà risposto con il testo contenuto nel messagio memorizzato all'indice indicato.

Listing 8: Implementazione della funzione per inviare il testo via SMS al numero indicato

```
// Invio SMS
void invioSMS(String numeroTelefono, String testo) {

    // Composizione del comando AT
    String comando = "AT+CMGS=\"";
    comando.concat(numeroTelefono);
    comando.concat("\"");

    // Invio del comando per l'invio di un SMS al modulo
    Serial.println(comando);

    // Invio al modulo del testo da inserire nel messaggio
    Serial.print(testo);

    // Invio al modulo del carattere ASCII corrispondente al comando Ctrl+Z per l'effettivo invio
        del messaggio
    Serial.write(26);

    return;
}
```

Quando nell'esecuzione del programma si richiama la funzione invioSMS, è necessario passare i parametri numeroTelefono e testo.
Il contenuto della variabile numeroTelefono viene utilizzata nella composizione del comando *AT* per l'invio del messaggio, mentre il contenuto della variabile testo viene inviato al modulo in un secondo momento. Per terminare il testo ed inviare effettivamente il messaggio si trasmette al modulo il valore "26" che viene trasmesso codificato in base al codice *ASCII* inviando così il carattere "SUB" (substitute), corrispondente alla combinazione di tasti "Ctrl+Z" della tastiera di un PC.

Listing 9: Implementazione della funzione per cancellare i messaggi in memoria

```
// Cancellazione SMS
void cancellazioneSMS() {

    // Invio del comando per la cancellazione di tutti gli SMS al modulo
    Serial.println("AT+CMGD=1,4");

    return;
}
```

Il comando *AT* utilizzato nella funzione cancellazioneSMS va a cancellare tutti i messaggi presenti in memoria. Il programma principale va quindi a richiamare tale funzione solamente una volta letti tutti i messaggi in memoria.

Connessione GPRS

Una normale connessione GPRS comprende le seguenti fasi:

- Verifica di corretta registrazione alla rete
- Inizializzazione della connessione GPRS
- Connessione al Server
- Utilizzo del metodo *GET* per la connessione HTTP
- Utilizzo del metodo *POST* per la connessione HTTP

Listing 10: Implementazione della funzione per verificare la registrazione alla rete cellulare

```
// Verifica registrazione alla rete cellulare
boolean verificaRegistrazione () {

  // Invio del comando per verificare la registrazione alla rete
  Serial.println("AT+CREG?");

  // Lettura della risposta
  String risposta = "";
  while(Serial.available()>0) { // Si verifica la presenza di comunicazione Seriale
  carattere = Serial.read(); // Si legge il carattere in arrivo
  risposta.concat(carattere); // Si compone la stringa ricevuta
  }

  // Elaborazione della stringa "risposta"
  int puntatorePiu = risposta.indexOf("+");
  // Estrazione carattere portante l'informazione desiderata
  risposta = risposta.substring(puntatorePiu);

  // Verifica dell'avvenuta registrazione confrontando la risposta con una stringa di riferimento
  boolean registrazione = risposta.startsWith("+CREG: 0,1");

  return registrazione;
}
```

Il comando *AT* utilizzato nella funzione verificaRegistrazione serve per richiedere al modulo informazioni riguardanti lo stato della registrazione alla rete cellulare.
Questa informazione è molto importante per sapere se si è registrati correttamente, oppure registrati in *roaming* oppure non registrati.
L'unica risposta con la quale si può procedere con la comunicazione internet è "+CREG:0,1" la quale indica che si è correttamente registrati alla rete cellulare proprietaria dell'operatore.
Un'altra risposta possibile è "+CREG:0,5" la quale indica che si è correttamente registrati ma in *roaming*. Si è scelto di non utilizzare la comunicazione internet in questa situazione perché la maggior parte degli operatori tariffano in modo differente tale utilizzo della rete. Quindi si è optato per il rinunciare alla connessione internet in questo caso, a garanzia dell'utente, per evitare costi telefonici eccessivi.

Listing 11: Implementazione della funzione per impostare l'APN per l'accesso alla rete

```
// Selezione APN corretto
String impostazioneAPN() {

    // Invio del comando per richiedere il codice ICCID della SIM inserita
    Serial.println("AT+CCID");

    // Lettura della risposta
    String  iccid = "";
    while(Serial.available()>0) { // Si verifica la presenza di comunicazione Seriale
    carattere = Serial.read(); // Si legge il carattere in arrivo
      iccid.concat(carattere); // Si compone la stringa ricevuta
      }

    // Estrazione del codice identificativo dell'operatore della SIM
    iccid = iccid.substring(6,8);

    // Ricerca dell'APN corretto tra quelli in memoria
    int indiceOperatore = -1;
    for (int k = 0; k < numeroOperatori; k++) {
        if (iccid.equals(codiciICCID[k])) {
            indiceOperatore = k;
        }
    }

    // Selezione nel vettore in memoria dell'APN dell'operatore corrispondente
    String apn = apnOperatori[indiceOperatore];

    return apn;
}
```

Il dispositivo necessita di una scheda SIM per poter operare correttamente. La stipulazione del contratto telefonico con l'operatore è a carico del cliente, il quale può scegliere tra i principali provider italiani.
Per l'accesso ad internet, ogni operatore utilizza un *APN* (Access Point Name) ben definito, ma differente da ogni altro operatore.
Il sistema deve essere in grado di riconoscere l'operatore che ha fornito la scheda SIM, per applicare l'APN corretto.
Ogni scheda SIM ha un codice univoco detto ICCID (Integrated Circuit Card IDentifier) che la identifica. Questo codice contiene all'interno due cifre (o tre in base all'operatore) che identificano in modo univoco l'operatore.
Si sono memorizzati i codici dei principali operatori nel vettore costante codiciICCID e nel vettore costante apnOperatori nelle rispettive posizioni gli APN. In questo modo, determinato l'indice di posizione del codice operatore nel primo vettore, nel secondo al medesimo indice si trova l'APN corretto.

Listing 12: Implementazione della funzione per settare le impostazioni per stabilire la connessione GPRS

```
// Inizializzazione della connessione alla rete cellulare
void inizializzazioneConnessione(String apn) {

    // Invio del comando per settare il tipo di connessione
    Serial.println("AT+SAPBR=3,1,\"Contype\",\"GPRS\"");

    // Composizione del comando AT
    String comando = "AT+SAPBR=3,1,\"APN\",\"";
    comando.concat(apn);
    comando.concat("\"");
    // Invio del comando per settare l'APN dell'operatore telefonico
    Serial.println(comando);

    // Invio del comando per settare l'user name per la connessione
    Serial.println("AT+SAPBR=3,1,\"USER\",\"\"");
    // Invio del comando per settare la password per la connessione
    Serial.println("AT+SAPBR=3,1,\"PWD\",\"\"");
    // Invio del comando per avviare la connessione
    Serial.println("AT+SAPBR=1,1");

    return;
}
```

La funzione inizializzazioneConnessione imposta tutti i parametri per il *Bearer Service*, necessari per tutte le funzioni basate sul Protocollo Internet.
Tra i parametri settati vi sono il tipo di connessione e l'APN. Il tipo di connessione viene impostato come GPRS, mentre l'APN che dipende dall'operatore telefonico della scheda SIM inserita nel dispositivo, viene passato come parametro della funzione.

Listing 13: Implementazione della funzione per impostare i parametri per stabilire la connessione HTTP

```
// Connessione al Server
void connessioneServer(String url) {

    // Invio del comando per inizializzare del servizio HTTP
    Serial.println("AT+HTTPINIT");

    // Invio del comando per utilizzare le impostazioni della connessione GPRS
    Serial.println("AT+HTTPPARA=\"CID\",1");

    // Composizione comando AT
    String comando = "AT+HTTPPARA=\"URL\",\"";
    comando.concat(url);
    comando.concat("\"");
    // Invio del comando per settare l'URL da contattare
    Serial.println(comando);

    return;
}
```

La funzione connessioneServer avvia il protocollo HTTP per la comunicazione. In questa funzione vengono solo settati i parametri della connessione GPRS e dell'URL della pagina alla quale si vuole connettere il dispositivo. La visita vera e propria avviene nelle funzioni connessioneGET o connessionePOST, a seconda del metodo che si vuole utilizzare.

Listing 14: Implementazione della funzione per stabilire la connessione HTTP con metodo GET

```
// Connessione HTTP con metodo GET
void connessioneGET() {

  // Invio del comando per effettuare la connessione HTTP con metodo GET all'URL impostato
  Serial.println("AT+HTTPACTION=0");

  // Invio del comando per leggere il corpo della risposta alla connessione HTTP
  Serial.println("AT+HTTPREAD");

   // Lettura della risposta
   String  corpoGET = "";
   while(Serial.available()>0) { // Si verifica la presenza di comunicazione Seriale
   carattere = Serial.read(); // Si legge il carattere in arrivo
     corpoGET.concat(carattere); // Si compone la stringa ricevuta
     }

   // Invio del comando per terminare la connessione HTTP
   Serial.println("AT+HTTPTERM");

   return corpoGET;
}
```

Nella funzione connessioneGET si utilizza il comando *AT* "AT+HTTPACTION=0" per avviare la connessione all'URL impostato. Il valore 0 indica che la connessione HTTP avviene con metodo GET.
Questo metodo risulta essere utile perché si possono trasmettere piccole informazioni inserendole direttamente nell'URL. Queste informazioni vengono poi elaborate con del codice PHP direttamente dal server, il quale invierà al dispositivo la pagina risultante, con le informazioni elaborate.
Per maggiori informazioni riguardanti l'implementazione della comunicazione su server, si faccia riferimento a [9].
La pagina web di risposta dal server viene scaricata e temporaneamente salvata nella memoria interna. Il comando *AT* AT+HTTPREAD richiede al modulo quanto memorizzato con il comando precedente, che viene trasmesso come una stringa, così che possa essere elaborata da *Arduino*.
Infine si termina il servizio HTTP con il comando AT+HTTPTERM.

Listing 15: Implementazione della funzione per stabilire la connessione HTTP con metodo POST

```
// Connessione HTTP con metodo POST
void connessionePOST(long numeroCaratteri, long posizionePrimoCarattere, long
    posizioneUltimoCarattere) {

  // Invio del comando per settare il tipo di contenuto per il metodo POST
  Serial.println("AT+HTTPPARA=\"CONTENT\",\"application/x-www-form-urlencoded\"");

  // Composizione del comando AT
  String comando = "AT+HTTPDATA=";
  comando.concat(String(numeroCaratteri));
  comando.concat(",120000");

  // Invio del comando per l'invio al al server del pacchetto dati
  Serial.println(comando);

  // Invio del pacchetto dati
  String carattere = "";
  for (long k = posizionePrimoCarattere; k <= posizioneUltimoCarattere; k++) {
    carattere = ""; // Pulizia variabile
    carattere.concat(leggiMemoria(k)); // Lettura del carattere in posizione k della memoria
        FLASH
    Serial.print(carattere); // Invio al modulo di un carattere alla volta
  }

 // Invio del comando per effettuare la connessione HTTP con metodo POST all'URL impostato
  Serial.println("AT+HTTPACTION=1");

  // Invio del comando per leggere il corpo della risposta alla connessione HTTP
  Serial.println("AT+HTTPREAD");

  // Lettura della risposta
  String corpoPOST = "";
  while(Serial.available()>0) { // Si verifica la presenza di comunicazione Seriale
  carattere = Serial.read(); // Si legge il carattere in arrivo
    corpoPOST.concat(carattere); // Si compone la stringa ricevuta
    }

  // Invio del comando per terminare la connessione HTTP
  Serial.println("AT+HTTPTERM");

  return corpoPOST;
}
```

Nella funzione connessionePOST si implementa una connessione HTTP con metodo POST. Si è rivelato necessario ricorrere a questo metodo in aggiunta al metodo GET per poter trasmettere al server pacchetti d'informazione che superano i limiti di dimensione caratteristici del metodo GET.
Prima di avviare la connessione, è necessario settare il tipo di contenuto nell'header HTTP. Tale configurazione comunica che il contenuto utilizza il metodo POST, e viene impostata mediante il comando *AT* "AT+HTTPPARA=\"CONTENT\",\"application/x-www-form-urlencoded\"".
Si procede inviando il comando *AT* AT+HTTPDATA per indicare al modulo quanti caratte-

ri sta per ricevere da inserire nel pacchetto per l'invio con metodo POST. Successivamente si inviano i dati al modulo, un carattere alla volta, per motivi di capacità dei microcontrollori e della comunicazione Seriale, che non riuscirebbero a gestire stringhe uniche di quella lunghezza. Il carattere viene letto direttamente dalla memoria FLASH e inviato sulla comunicazione Seriale. Una volta inviato al modulo l'intera informazione, si utilizza il comando *AT* "AT+HTTPACTION=1" per avviare la connessione all'URL impostato. Il valore 1 indica che la connessione HTTP avviene con metodo POST.
La principale situazione in cui è necessario ricorrere al metodo POST nell'applicazione qui presentata, è quando si deve inviare al server la registrazione di uno o più viaggi. Infatti l'invio al server dei viaggi registrati avviene solo nel caso sia il cliente a richiederlo, quindi è servito porsi nel caso peggiore, ovvero nel caso di più viaggi differenti memorizzati da sincronizzare.
La pagina web di risposta dal server viene scaricata e temporaneamente salvata nella memoria interna.
Come fatto per la funzione connessioneGET, si invia il comando *AT* AT+HTTPREAD per richiedere al modulo quanto memorizzato con il comando precedente, che viene trasmesso come una stringa, così che possa essere elaborata da *Arduino*.
Infine si termina il servizio HTTP con il comando AT+HTTPTERM.

5.2.3 Comunicazione GPS

Il modulo *SIM908* fornisce una soluzione GPS adatta per applicazioni portatili.
Supporta sia operazioni per un utilizzo in dispositivi di consumo di navigazione portatili che in altri sistemi autonomi di navigazione.

Come già visto per altre funzioni del modulo *SIM908*, anche per il posizionamento si ricorrono agli *AT Commands*.

Si riporta di seguito in Listing 16 un esempio di implementazione su *Arduino* del comando *AT* che richiede al modulo *SIM908* la posizione:

Listing 16: Implementazione della funzione per ottenere informazioni GPS

```
// Richiesta informazioni GPS
String informazioniGPS(){
    // Invio del comando
    Serial.println("AT+CGPSINF=32"); // Invio del comando AT Per richiesta poszione

    // Ricezione della risposta
    while(Serial.available()>0) { // Si verifica la presenza di comunicazione Seriale
        carattere = Serial.read(); // Si legge il carattere in arrivo
        posizione.concat(carattere); // Si compone la stringa ricevuta
    }
    return posizione;
}
```

Il codice AT+CGPSINF=32 richiede al modulo *SIM908* le informazioni di geo-localizzazione. Il valore "32" indica che il modulo invia al microcontrollore l'informazione GPS secondo il protocollo *RMC*, dello standard *NMEA 0183*. Questo standard fornisce le specifiche elettriche e d'informazione per comunicazioni tra dispositivi elettronici marini. Tra questi rientrano anche i dispositivi GPS, nonostante stiano prendendo sempre più posto nella vita di tutti i giorni. Questo standard è definito e controllato da *National Marine Electroinics Association* (NMEA). Nello standard *NMEA* sono definiti diversi formati, per l'informazione. Per l'applicazione è stato scelto il formato *RMC* (Recommended minimum specific GPS/Transit data). Questo

formato risulta il più ricco d'informazione, come si può vedere in Tabella 4 dove sono riportate le parti che compongono la risposta in tale formato.

Name	Description
Message ID	TMC protocol header
UTC Time	hhmmss.sss
Status	A=data valid or V=data not valid
Latitude	ddmm.mmmmmm
N/S Indicator	N=north or S=south
Longitude	dddmm.mmmmmm
E/W Indicator	E=east or W=west
Speed Over Ground	
Course Over Ground	TRUE
Date	ddmmyy
E/W Magnetic Variation	E=east or W=west
East/West Indicator	E=east
Mode	A=Autonomous, D=DGPS, E=DR, N=Output Data Not Valid R=Coarse Position
Checksum	
<CR><LF>	End of message termination

Tabella 4: Recommended minimum specific GPS/Transit data

Un esempio di risposta che si potrebbe trovare nella stringa posizione è:
32,161450.000,A,4524.545040,N,1153.643960,E,0.00,297.43,140715,,E,A
Si osserva che ogni informazione è separata dalla vicina con una ",". Questo si presta all'elaborazione della stringa basata su tale carattere per l'estrazione delle informazioni necessarie. Come ci si aspetta, la risposta segue quanto dichiarato nello standard riportato in Tabella 4. Tuttavia il formato delle informazioni "Latitudine" e "Longitudine" non è consono per l'utilizzo all'interno dell'applicazione. Infatti si ha una scrittura del tipo "ddmm.mmmmmm" e "dddmm.mmmmmm", rispettivamente, dove "dd" e "ddd" corrispondono ai gradi e "mm.mmmmmm" ai primi. Il formato più adatto per l'applicazione è "dd.dddddd" e "ddd.dddddd", ovvero i primi riportati come parte decimale del grado. Si ricorre quindi ad una conversione. Una volta trasformate le informazioni da stringhe a valori numerici, le formule che si applicano sono le seguenti:

$$\text{latitudine} = \text{gradi} + \frac{\text{primi}}{60}$$
$$\text{longitudine} = \text{gradi} + \frac{\text{primi}}{60}$$

Quindi le informazioni d'interesse latitudine e longitudine che si ottengono nel caso in esempio, in seguito a questa semplice trasformazione valgono 45.409084 e 11.894066.

5.2.4 Gestione delle richieste

Un problema riscontrato è legato al fatto che la comunicazione attraverso il modulo *SIM808* può richiedere tempi elevati, paragonabili a quelli dell'intero sistema. A causa di questo si otterranno cicli con tempi molto differenti l'uno dall'altro, con conseguente "arresto" del sistema in attesa della risposta.
Per ovviare a questo problema, l'automa implementato non invia direttamente gli *AT Commands* al modulo, ma è stata costruita una coda circolare con modalità *FIFO* dove l'automa inserisce le richieste per il modulo. È stato costruito un secondo automa a stati finiti deterministico, indipendente dal primo, il quale legge l'elemento in coda e invia la richiesta al *SIM808*. Il difetto di questa soluzione risulta essere che ogni ciclo il sistema deve seguire il funzionamento di due automi distinti.
Per contro i pregi sono:

- separando funzionamento principale da comandi modulo, non si rischia di sovrapporre le richieste;
- per alcune richieste per il modulo *SIM808* sono necessari diversi *AT Commands*. Avendo un automa che gestisce solo le richieste, è possibile suddividere questi comandi su cicli differenti, con bassa incidenza sul tempo di ciclo.

In Listing 17 si riporta un esempio dell'implementazione di questa coda di richieste:

Listing 17: Implementazione della funzione per gestire la coda delle richieste

```
// Generazione della coda FIFO
String codaRichieste[50]; // La coda e' un semplice vettore di 50 elementi String

// Funzione per l'inserimento di una richiesta
void inserimentoRichiesta() {

    // Aggiunta della richiesta
    codaRichieste[puntatoreRichieste] = richiesta;
    // Aggiornamento puntatore inserimento richieste in modo circolare
    puntatoreRichieste = (puntatoreRichieste+1)%50;

    return;
}

// Funzione per l'esecuzione della richiesta
void exeRich(){
    // Verifica che ci sia almeno una richiesta nella coda
    if(puntatoreEsecuzioni!=puntatoreRichieste){

    // Verifica della richiesta inserita
    if(codaRichieste[puntatoreEsecuzioni].equals(richiestaRing)){
        // Attivazione della funzione di invio di un SMS
        invioSMS = true;
        // Cancellazione della richiesta dalla coda
        codaRichieste[puntatoreEsecuzioni] = "";
        //Aggiornamento puntatore inserimento richieste in modo circolare
        puntatoreEsecuzioni = (puntatoreEsecuzioni+1)%50;
    }
}
```

6 Verifica di funzionamento

In seguito al montaggio e alla programmazione del prototipo, si sono eseguite varie fasi di test. Le singole prove sono indirizzate a verificare la corretta esecuzione di tutte le funzionalità del prototipo.
La verifica è suddivisa in due fasi:

- Verifica effettuata direttamente sul dispositivo
- Verifica effettuata attraverso interfaccia utente

Di seguito vengono esposti i metodi utilizzati ed analizzati i risultati.

6.1 Debug del dispositivo

La fase di verifica effettuata direttamente sul dispositivo riguarda la possibilità di monitorare il funzionamento tramite collegamento diretto al microcontrollore. *Arduino DUE*, infatti, permette di instaurare una comunicazione Seriale con un PC attraverso il sistema di programmazione dello stesso.
Tale comunicazione è sfruttabile al fine di ottenere un sistema di debug real-time per verificare il corretto funzionamento del dispositivo. Di seguito sono riportati due tipi di verifiche che sono state effettuate grazie a tale comunicazione.

6.1.1 Lettura Seriale

Attraverso il codice `Serial.print(`*Dato da stampare*`)`, messo a disposizione dal linguaggio di programmazione di *Arduino*, è possibile inviare una qualsiasi stringa o il valore contenuto in una variabile.
Questo comando si rivela utile per ottenere un'informazione real-time del funzionamento di Arduino, infatti ponendolo in punti strategici del programma, è possibile monitorare istante per istante le azioni del microcontrollore e i valori memorizzati nelle variabili.

Sul PC si ha necessità di interpretare i dati trasmessi in un'interfaccia user-friendly. Al riguardo, l'ambiente di sviluppo integrato (IDE) di *Arduino* mette a disposizione un *Monitor Seriale* che permette di visualizzare in formato letterale i dati ricevuti.
In Figura 17 è riportato un'acquisizione dallo schermo del PC durante il monitoraggio del dispositivo da Monitor Seriale di *Arduino*.

Dalla Figura 17 si può vedere che le potenzialità di questo metodo di debug sono molteplici in fase di test.
Tuttavia, il Monitor Seriale messo a disposizione ha delle limitazioni che potrebbero creare dei problemi:

- Il sistema riserva una memoria limitata al programma, quindi non tutto lo storico risulta disponibile, soprattutto in caso di lunghe sessioni di test, che purtroppo sono le più indicate per il dispositivo qui presentato.
- Il Monitor Seriale non permette di copiare il testo ricevuto da *Arduino*, impedendo quindi il salvataggio dello stesso per una analisi a posteriori.

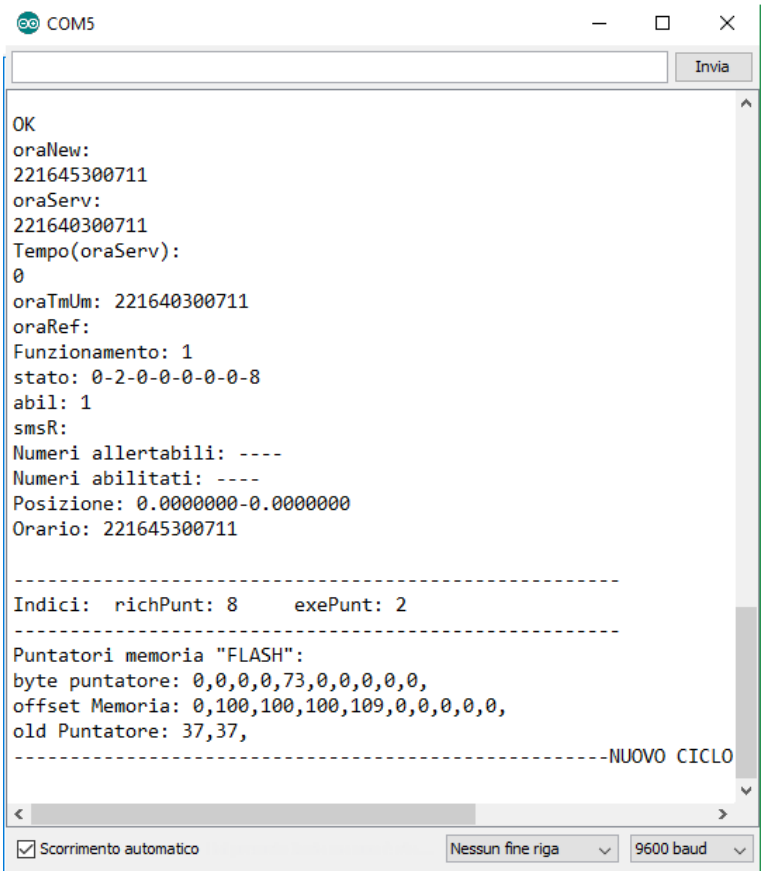


Figura 17: Monitor Seriale integrato nell'IDE di Arduino

Queste limitazioni hanno spinto a cercare una soluzione alternativa, la quale ha trovato risposta nel programma *Processing*. Questo programma open-source, è molto flessibile, con molti strumenti e funzionalità, che vanno ben oltre la gestione della comunicazione Seriale.
Per maggiori informazioni si faccia riferimento alla pagina web del programma [8].

Nonostante le molte funzionalità, *Processing* presenta un linguaggio relativamente semplice e di immediata comprensione per chiunque abbia una sufficiente conoscenza dei più comuni linguaggi di programmazione (C, C++, Java).

L'ambiente di sviluppo di *Processing*, che può ricordare quello di *Arduino*, integra un monitor, sul quale il codice, per mezzo del comando output.print(*Dato da stampare*), può stampare delle informazioni.
Il risultato ottenuto con tale comando è del tutto analogo a quanto si ottiene con *Arduino* con l'utilizzo del comando Serial.print(*Dato da stampare*).

Si riporta in Listing 18 il codice *Processing* per la lettura della comunicazione Seriale proveniente da *Arduino* e la scrittura della stessa su file di testo.

Listing 18: Codice Processing per lettura da porta Seriale e scrittura su file di testo

```
// Scrittura su file da Arduino
import processing.serial.*;
Serial mySerial;
PrintWriter output;

void setup() {
  // Settaggio della porta seriale di Arduino e del bitrate
  mySerial = new Serial( this, Serial.list()[1], 9600 );
  // Settaggio del buffer seriale fino al carattere "new line"
  mySerial.bufferUntil('\n');

  // Apertura del file "dati.txt" per poter permettere operazioni di lettura, scrittura e
      cancellazione dello stesso
  output = createWriter( "dati.txt" );
}

// Inizio del programma
void draw() {

  // Verifica di dati nel buffer seriale
  if (mySerial.available() > 0 ) {

    // Lettura di una riga di dati del buffer seriale
    String value = mySerial.readStringUntil('\n');

    // Verifica che la riga letta non sia vuota
    if ( value != null ) {

      // Scrittura sul file "dati.txt" del valore letto dal buffer seriale
      output.print( value );
      // Scrittura sul terminale del valore letto dal buffer seriale
      print(value);
    }
  }
}
```

Il funzionamento del codice riportato in Listing 18 è particolarmente semplice. Dopo una parte di setup dove vengono inizializzati la comunicazione seriale e il file di testo dove scrivere, il programma, quando trova dei dati nel buffer, li legge e li scrive nel file di testo e nel terminale integrato nell'IDE *Processing*. La combinazione di "output.print(value)" e "print(value)" permette di salvare in modo automatico la comunicazione Seriale nel file di testo per un controllo a posteriori, sul quale è possibile fare qualsiasi operazione di copia. Inoltre è possibile effettuare un monitoraggio real-time.
In Figura 18 è riportata un'acquisizione dallo schermo del PC dove si può vedere quanto appena spiegato. A sinistra si può vedere l'IDE di *Processing* con il monitor dove si può leggere quanto riceve *Processing* dalla comunicazione seriale in modo real-time, mentre a destra è riportato il file di testo che si ottiene una volta che il test è ultimato il monitoraggio è stato arrestato.

Grazie al metodo di verifica attraverso la comunicazione Seriale si è riusciti a risolvere molti problemi di funzionamento riscontrati in fase di programmazione del prototipo. Inoltre si è potuto stimare la durata media del tempo di ciclo. Infatti ad ogni ciclo viene aggiornato l'orario

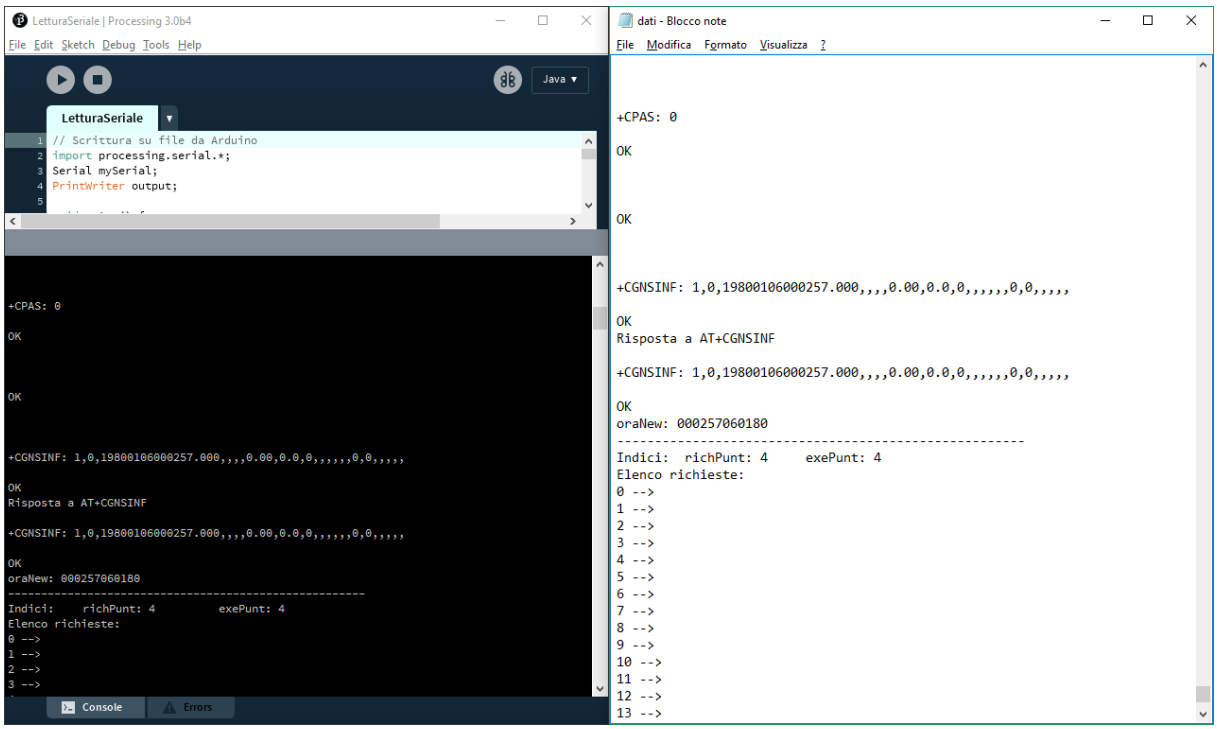


Figura 18: A sinistra l'IDE di Processing con relativo monitor, a destra il file di testo, risultato della scrittura della comunicazione seriale

del sistema sulla base delle informazioni ricevute dalla connessione GPS. Per determinare il tempo di ciclo in modo approssimato è sufficiente leggere l'orario di due cicli successivi e farne la differenza:

Listing 19: Esempio di due cicli consecutivi acquisiti con comunicazione Seriale

```
1  ----------  NUOVO CICLO----------
2
3  32,145313.000,A,4524.545040,N,1153.643960,E,0.00,0.00,180715,,E,A
4  OK
5  Funzionamento: 1
6  stato: 0-2-0-0-0-0-0-1
7  Posizione: 45.409084-11.894066
8  Orario: 145313180715
9  Indici: richPunt: 22 exePunt: 10
10
11 ----------  NUOVO CICLO----------
12
13 32,145316.000,A,4524.545040,N,1153.643960,E,0.00,0.00,180715,,E,A
14 OK
15 Funzionamento: 1
16 stato: 0-2-0-0-0-0-0-7
17 Posizione: 45.409084-11.894066
18 Orario: 145316180715
19 Indici: richPunt: 24 exePunt: 12
```

In Listing 19 sono riportati due cicli consecutivi acquisiti durante una fase di test. Il parametro d'interesse per determinare il tempo di ciclo sono i valori che assume la riga "Orario", ovvero 145313180715 prima e 145316180715 dopo. In questi due valori ogni coppia di valori indica un'informazione, in particolar modo la terza coppia (13 nel primo e 16 nel secondo) indica

i secondi. Nell'esempio qui presentato il tempo di ciclo è circa 3 s. Ovviamente misurare un tempo in secondi quando la frequenza di calcolo del microcontrollore è di 84 MHz, introduce errori rilevanti. Per avere un valore preciso, bisognerebbe ricorrere a strumenti differenti, più sensibili. Tuttavia, come già detto, è inutile voler misurare in modo più accurato il tempo di ciclo, quando questo dipende fortemente dai mezzi di comunicazione che utilizza, i quali hanno tempi caratteristici ben più lunghi.

6.1.2 Lettura memoria FLASH

Il sistema di verifica presentato nella sezione precedente permette un monitoraggio real-time del funzionamento del dispositivo. Tuttavia è stato implementato un secondo sistema di verifica collegando direttamente il dispositivo al PC, in quanto non sempre vi è la possibilità di un monitoraggio effettuato in modo continuativo, soprattutto dei sui dispositivi montati a bordo dell'imbarcazione.
Tale sistema si basa sulla lettura della memoria FLASH che è stata prevista nel dispositivo. All'interno della memoria vengono salvati alcuni valori che devono essere mantenuti anche in caso di mancata alimentazione, come per esempio i *numeri abilitati* e i *numeri allertabili*.
Oltre a questi dati, nella memoria FLASH si possono trovare uno storico contenente gli stati dell'imbarcazione memorizzati durante il normale funzionamento, che non sono ancora stati inviati correttamente al server. Combinando le informazioni dello storico presenti sul server con quelle presenti nella memoria FLASH si ottiene un quadro completo delle operazioni del dispositivo, permettendo così l'analisi, la rilevazione e la correzione di eventuali errori o di problemi che potrebbero presentarsi.

Tuttavia, il sistema di verifica implementato, a differenza da quanto mostrato nella sezione precedente, non è utilizzabile direttamente sul dispositivo durante il normale funzionamento. Infatti non da motivo di inviare via comunicazione Seriale il contenuto della memoria durante ogni ciclo, poiché tale operazione richiederebbe molto tempo. Si è quindi scelto di rendere questa verifica un metodo di controllo a posteriori.
Questo metodo è riassumibile nei seguenti passaggi:

1. Prelevare il dispositivo dall'imbarcazione
2. Sovrascrivere il programma del normale funzionamento memorizzato nel microcontrollore con quello realizzato per la lettura della memoria
3. Collegare il dispositivo al PC per la comunicazione Seriale

In Listing 20 è riportato il codice che implementa la funzione per la lettura della memoria FLASH.

Listing 20: Implementazione della funzione per la lettura della memoria FLASH

```
SPIFlash flash(FLASH_SS, 0xEF30);
int blocco = 1;
String concatIn="";

void setup(){
  // Inizializzazione comunicazione Seriale
  Serial.begin(9600);
  while(!Serial);
  // Inizializzazione comunicazione SPI per memoria FLASH
  flash.initialize();
}

void loop(){
  // Metodo per attendere un input da PC
  while(!Serial.available()>0){}
  // Inizializzazione stringa
  concatIn = "";
  // Verifica presenza dati nel buffer Seriale
  while(Serial.available()>0){
    // Lettura dell'informazione su comunicazione Seriale, un carattere alla volta
    char input = Serial.read();
    concatIn.concat(input);
  }

  // Ciclo for per la lettura dei primi 10 blocchi della memoria FLASH
  for(int l=1; l<11; l++){
    blocco = l;
    // Scrittura su comunicazione Seriale per interfaccia utente
    Serial.print("Lettura blocco numero: ");
    Serial.println(blocco);

    String lettura = "";
    // Ciclo for per la lettura del blocco corrispondente
    for(long k=blocco*4096L; k<(blocco+1)*4096L-1; k++){
      // Lettura della memoria FLASH, un carattere alla volta
      lettura.concat((char)flash.readByte(k));
    }
    // Invio per mezzo di comunicazione Seriale del contenuto della memoria letto
    Serial.println(lettura);
  }

  // Metodo per la cancellazione della memoria, con conseguente ripristino dei dati di fabbrica
  // Riconoscimento della stringa "Cancella" trasmesso via Seriale all'inizio della procedura
  if(concatIn.equals("Cancella")){
    // Interfaccia utente per segnalare la cancellazione dei dati in memoria
    Serial.println("Cancellazione");
    // Funzione per la cancellazione di tutti i blocchi della memoria FLASH
    flash.chipErase();
    // Interfaccia utente per segnalare il completamento dell'azione di cancellazione
    Serial.println("finito");
  }
}
```

In Listing 20, si instaura una comunicazione Seriale bidirezionale tra *Arduino* e il PC. Infatti all'inizio del "void loop()", il programma attende di ricevere dal PC un qualsiasi dato. Questo dato viene poi considerato solamente alla fine del processo, e assume valore solo se la stringa passata è uguale a "Cancella". Il programma, una volta ricevuto un dato dalla comunicazione Seriale, qualunque esso sia, procede con la lettura dei primi 10 blocchi di memoria. Si è limitato a questi in quanto sono i blocchi più ricchi d'informazione, inoltre si è mantenuta la suddivisione in blocchi per una maggior facilità di analisi del risultato prodotto.
Attraverso la funzione "readByte(k)" implementata dalla libreria apposita per la gestione delle memorie FLASH con comunicazione SPI, è possibile leggere un byte alla volta, indicando il numero della cella di memoria da leggere. Il tutto, una volta composto in una frase unica, viene inviato al PC attraverso la comunicazione Seriale.
Infine si verifica se la stringa ricevuta all'inizio è uguale a "Cancella". In caso affermativo si procede con la cancellazione di tutti i dati presenti nella memoria, in caso contrario il programma passa oltre, ultimando la procedura. La cancellazione avviene attraverso la funzione "chipErase()" messa a disposizione dalla libreria. In seguito a questo comando la memoria sarà completamente pulita, come quando il dispositivo è stato prodotto. Per poter ricominciare ad utilizzare il dispositivo, oltre a caricare nuovamente il programma del normale funzionamento, sarà necessario eseguire nuovamente la procedura di inizializzazione, cosa non necessaria in caso la memoria non venisse cancellata.

6.2 Test allarmi

Fra le prove effettuate si sono eseguiti anche test di normale funzionamento. In particolare si sono verificati il corretto funzionamento della segnalazione degli allarmi, della registrazione dei viaggi e del controllo dei dispositivi da remoto.

Per effettuare questo test si è preparato un banco di prova che simulasse tutti i collegamenti che si configurano durante l'installazione nell'imbarcazione. Si sono quindi collegati i cavi di alimentazione, tutti i sensori e i trasduttori, gli attuatori ed infine le antenne.
Per il funzionamento non viene richiesto altro, ma in fase di test si è comunque combinata la lettura Seriale descritta in precedenza per monitorare al meglio il dispositivo e verificarne il corretto funzionamento.

Si presenta quindi la sequenza di operazioni eseguite per l'analisi, in modo tale da renderla ripetibile:

1. Avvio del dispositivo e attesa di completa inizializzazione
2. Attivazione del Funzionamento Protetto
3. Generazione di un evento anomalo
4. Monitoraggio del dispositivo da *Lettura Seriale* e verifica di notifica su smartphone
5. (Opzionale) Generazione di ulteriori eventi anomali e relativo monitoraggio
6. Disattivazione del funzionamento Allarme da smartphone

Di seguito si riportano alcuni estratti del monitoraggio compiuto con il metodo di lettura Seriale descritto in precedenza. Per comodità, vengono riportati solo i cicli contenenti informazioni rilevanti al fine della verifica del funzionamento.

Listing 21: Monitoraggio del dispositivo durante test di notifica allarme

```
    ------------------------     NUOVO CICLO ------------------------
--textHTTPREAD: ********************************ok scaricato correttamente
--Ricevuta richiesta Funzionamento Protetto
Funzionamento: Libero
stato: 0-2-0-0-0-0-0-10
Orario: 115546200715
Indici:    richPunt: 10    exePunt: 8
-
-
-
    ------------------------     NUOVO CICLO ------------------------
Funzionamento: Protetto
stato: 0-0-0-0-0-0-0-9
Orario: 115603200715
Indici:    richPunt: 12    exePunt: 10
    ------------------------     NUOVO CICLO ------------------------
--Rilevato evento anomalo
Funzionamento: Protetto
stato: 0-0-5-0-0-0-0-10
Orario: 115606200715
Indici:    richPunt: 14    exePunt: 10
    ------------------------     NUOVO CICLO ------------------------
Funzionamento: Allarme
stato: 0-0-0-0-0-0-0-1
Orario: 115613200715
Indici:    richPunt: 14    exePunt: 10
-
-
-
    ------------------------     NUOVO CICLO ------------------------
--Inizio sequenza di notifica
--Composizione chiamata: ATD**********;
Funzionamento: Allarme
stato: 0-0-0-2-0-0-0-3
Orario: 115625200715
Indici:    richPunt: 7     exePunt: 1
-
-
-
    ------------------------     NUOVO CICLO ------------------------
--Invio SMS notifica allarme
Funzionamento: Allarme
stato: 0-0-0-2-0-0-0-1
Orario: 115649200715
Indici:    richPunt: 7     exePunt: 4
-
-
-
    ------------------------     NUOVO CICLO ------------------------
--Ricevuto SMS da numero abilitato: COD08:0
Funzionamento: Allarme
stato: 0-0-0-5-0-0-0-1
```

```
53 --smsR: COD08:0
54 Orario: 115759200715
55 Indici:    richPunt: 7    exePunt: 7
56    ------------------------    NUOVO CICLO ------------------------
57 Funzionamento: Libero
58 stato: 0-0-0-0-0-0-0-5
59 Orario: 115802200715
60 Indici:    richPunt: 12    exePunt: 10
```

In Listing 21 le informazioni sensibili (quali numeri di telefono o codici privati) sono state sostituite da "$\star$", mentre la sequenza di tre "-" in verticale, indica che sono stati omessi dei cicli privi di interesse.

Le informazioni che si sono ricavate da questo test sono:

- Il corretto funzionamento del dispositivo, il quale una volta rilevati uno o più eventi anomali, provvede a notificarli al cliente, e la possibilità da parte di quest'ultimo di disattivare l'allarme da remoto.
- Tempo di attivazione del Funzionamento Protetto: è il tempo che intercorre tra l'istante in cui arriva l'informazione di attivazione del funzionamento e l'instante in cui il sistema entra effettivamente in funzionamento Protetto. Lo si ottiene facilmente, calcolando il temo intercorso tra il primo ciclo dove compare la scritta "Ricevuta richiesta Funzionamento Protetto" ed il primo ciclo con "Funzionamento: Protetto". In questo caso il tempo misurato è all'incirca 15 secondi.
- Tempo di reazione ad un evento anomalo: è il tempo che intercorre tra l'istante in cui viene rilevato un evento anomalo e l'istante in cui il dispositivo passa in funzionamento Allarme. In Listing 21 lo si può determinare calcolando il tempo intercorso tra il ciclo dove compare "--Rilevato evento anomalo" e il primo ciclo con "Funzionamento: Allarme". In questo caso il tempo è circa 7 secondi.
- Tempo di notifica di un evento anomalo: è il tempo che intercorre tra l'istante in cui viene rilevato un evento anomalo e l'istante in cui il dispositivo inizia la fase di notifica, ovvero quando viene avviata la telefonata. In Listing 21 lo si può determinare calcolando il tempo intercorso tra il ciclo dove compare "--Rilevato evento anomalo" e il dove compare "Inizio sequenza di notifica". In questo caso il tempo è circa 19 secondi.
- Tempo di disattivazione dell'allarme: è il tempo che intercorre tra l'istante in cui arriva l'informazione di disattivazione dell'allarme e l'instante in cui il sistema esce effettivamente dal funzionamento Allarme e Protetto. In Listing 21 lo si può determinare calcolando il tempo intercorso tra il ciclo dove compare "--Ricevuto SMS da numero abilitato: COD08:0" e il primo ciclo con "Funzionamento: Libero". In questo caso tempo è di circa 3 secondi.

Come si può vedere, tutti i tempi misurati sono in buon accordo con le specifiche di progetto (che prevedevano tempi dell'ordine di alcune decine di secondi).

6.3 Test controllo remoto

Come fatto per il test degli allarmi, anche in questa verifica si è utilizzato il banco di prova che simulasse tutti i collegamenti che si configurano durante l'installazione nell'imbarcazione.

Si presenta quindi la sequenza di operazioni eseguite per l'analisi, in modo tale da renderla ripetibile:

1. Avvio del dispositivo e attesa di completa inizializzazione
2. Invio da smartphone della richiesta di accensione dell'attuatore scelto
3. Attesa della effettiva accensione
4. Invio da smartphone della richiesta di spegnimento dell'attuatore
5. (Opzionale) Invio da smartphone di altre richieste di controllo

In questo caso non si riportano i risultati della Lettura Seriale, perché di scarso interesse, in quanto l'unica informazione che se ne ricava in modo diretto risulta essere il corretto funzionamento e i cicli impiegati tra l'arrivo dell'informazione e l'effettiva esecuzione.
I risultati ottenuti sono quindi:

- La verifica del corretto funzionamento del dispositivo, il quale, una volta ricevuta la richiesta di attivazione/disattivazione di un attuatore, la applica e risponde al mittente sul medesimo canale di comunicazione
- Il tempo di attuazione, ovvero il tempo che intercorre tra quando arriva l'informazione al dispositivo e l'istante in cui questa viene effettivamente applicata è di circa $2 \div 3$ cicli, quindi pari ad un tempo inferiore ai 10 secondi

6.4 Test viaggi

Tra le funzionalità che si sono testate, particolare importanza è stata data alla registrazione dei viaggi compiuti.

Per effettuare questa verifica non occorre il banco di prova presentato per i test precedenti. Essendo un test da effettuare trasportando il dispositivo, si è preferito ridurre i collegamenti alle sole antenne, sfruttando la batteria tampone interna.

La lettura Seriale in questo caso risulta scomoda, proprio per una questione di trasportabilità del sistema. Di conseguenza, per questo test si è utilizzato lo smartphone per la verifica del corretto funzionamento e successivamente la lettura della memoria FLASH per la ricerca di errori.

Si presenta quindi la sequenza di operazioni eseguite per l'analisi, in modo tale da renderla ripetibile:

1. Impostazione del funzionamento Libero
2. Abilitazione della funzione di tracking
3. Inizio spostamento con velocità superiore a 1 nodo
4. Esecuzione di un tragitto di distanza considerevole

5. Termine spostamento con velocità inferiore a 1 nodo
6. Spegnimento dispositivo
7. Verifica Lettura della memoria FLASH
8. Riaccensione dispositivo
9. Richiesta di upload su server del viaggio da parte dello smartphone
10. Verifica sullo smartphone della corretta registrazione del viaggio

Dall'analisi della memoria FLASH, si possono ottenere tutte le coordinate salvate durante il test. Risulta quindi utile in caso si siano verificati errori. Molto più interessante è il risultato visualizzato sullo schermo dello smartphone, il quale è stato riportato in Figura 19.

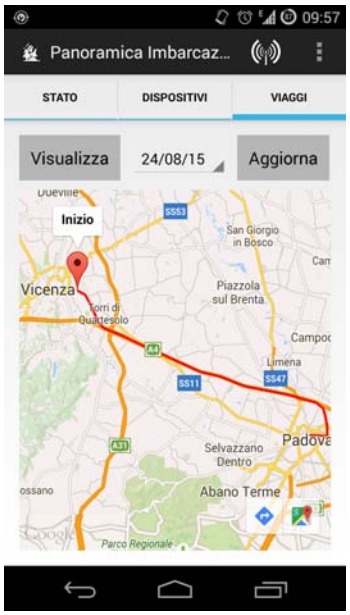


Figura 19: Risultato del Test viaggi

Si riassumono quindi le informazioni che si sono ricavate in questa verifica:

- Il corretto funzionamento del dispositivo, il quale è in grado di rilevare la propria posizione e memorizzarla per poi renderla disponibile allo smartphone del cliente per mezzo del server.
- Il dispositivo rispetta gli intervalli di acquisizione e le soglie di velocità impostati.
- La comunicazione HTTP con metodo POST per l'invio al server dei dati riguardanti i viaggi si è rivelata corretta.

7 Revisione progetto prototipo

Tutte le prove e le verifiche fino a questo punto riportate, sono state effettuate con il prototipo presentato in Sezione 4, e del quale si riporta un'immagine in Figura 20.

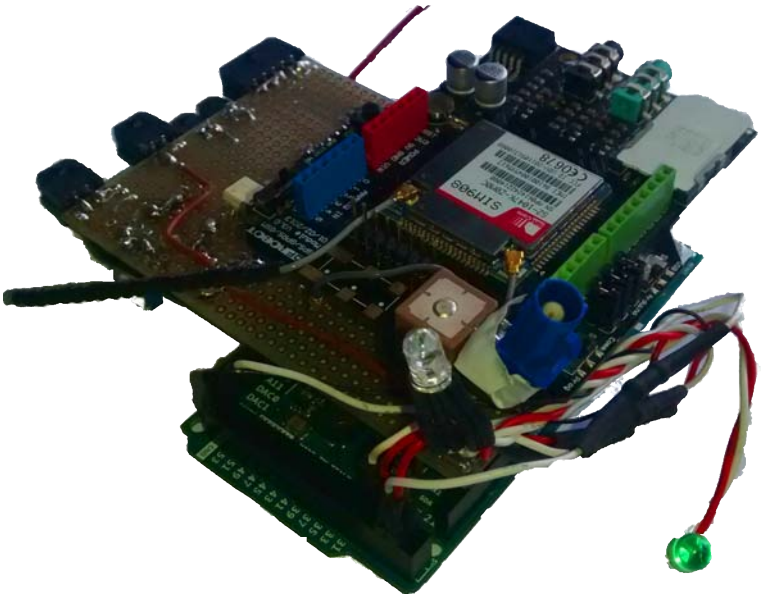


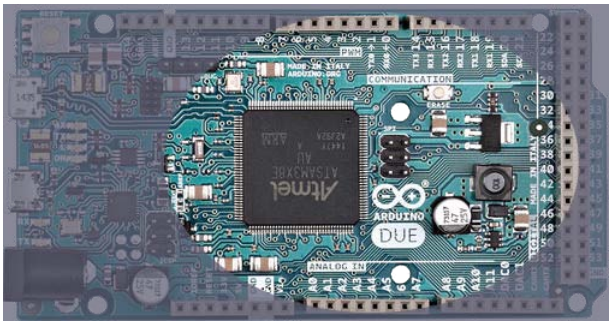
Figura 20: Primo prototipo

L'obbiettivo di questo progetto tuttavia è creare una versione del dispositivo che possa essere messa in produzione e commercializzato.
A tal fine si deve quindi raggiungere una struttura adeguata, che nel caso qui presentato si traduce in una scheda unica che riunisca tutti i componenti, possibilmente ricorrendo all'utilizzo di componenti a montaggio superficiale.
Prima di giungere a questa, si è preferito porre uno step intermedio, nel quale si sono riprogettate le schede presentate in Sezione 4 mantenendole separate, in modo da permettere una più facile verifica del funzionamento delle singole parti.
Le schede sono state inoltre modificate, adattandole all'applicazione qui presentata.

7.1 Arduino

Arduino è un progetto *Open-source*, del quale si possono ottenere tutti gli schemi elettrici e i firmware.
Nel caso in esame, si utilizza un *Arduino DUE*, il quale mette a disposizione molti più pin rispetto a quelli richiesti dall'applicazione, occupando quindi più spazio.
In un'ottica del risparmio in termini economici, ma soprattutto di spazio, si è scelto di separare il microcontrollore dalla parte necessaria per programmare. Questa scelta è motivata anche per motivi di sicurezza, ovvero, separando la parte di programmazione/interfaccia USB, si rende più difficoltoso ad un utente esterno raggiungere il codice del programma memorizzato all'interno del microcontrollore stesso.

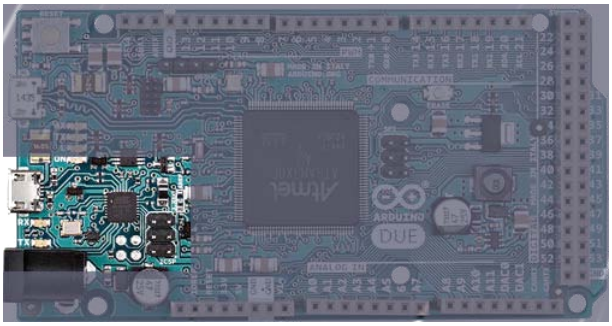
Dopo un'attenta analisi, si è realizzato un microcontrollore con tutte e sole le funzionalità di *Arduino DUE* che sono state utilizzate nel progetto. Inoltre si è realizzata una scheda per la programmazione/interfaccia USB ad hoc. Grazie a questa è possibile far riconoscere al PC il dispositivo come *Arduino DUE* e quindi programmarlo come tale.
Il risultato della revisione è riportato nelle Figure 21 e 22.



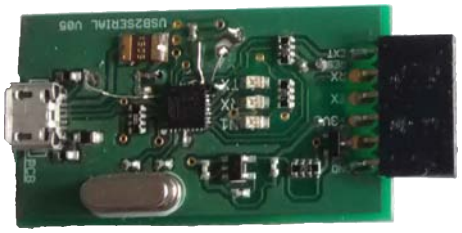
(a) Focus microcontrollore *Arduino DUE*

(b) Microcontrollore realizzato per il progetto

Figura 21: Revisione del microcontrollore di *Arduino DUE*



(a) Focus programmatore *Arduino DUE*



(b) Programmatore realizzato per il progetto

Figura 22: Revisione del programmatore di *Arduino DUE*

7.2 Shield GSM/GPRS/GPS

In fase di riprogettazione dello shield GSM/GPRS/GPS ci si è scontrati con il problema che il modulo *SIM908*, sul quale si basa lo shield presentato in Sezione 4.2 è uscito di produzione.
Questo è un problema da non sottovalutare, in quanto il progetto è destinato a diventare un prodotto commerciale, e non si può basarsi su prodotti che potrebbero non essere più reperibili.
Si è quindi ricercato un modulo che potesse sostituire l'attuale.
Per la ricerca si è scelto di rimanere tra i prodotti di *SIMCom*, in modo da trovare una maggior compatibilità con il codice già sviluppato. Si è quindi scelto il modulo *SIM808* prodotto dalla *SIMCom*, il quale racchiude tutte le funzionalità del precedente *SIM908* e ne aggiunge delle altre quali la connettività bluetooth, ottima in previsione di futuri sviluppi.
Per maggiori informazioni si faccia riferimento alla pagina web del prodotto [3].
Come per il modulo *SIM908*, anche il *SIM808* comunica con *Arduino* attraverso la comunicazione Seriale.

7.2.1 Hardware

Il modulo *SIM808* è un prodotto nuovo della *SIMCom*, quindi ad oggi non si trovano shield prefabbricati, basati su questo modulo.
Si è quindi provveduto alla realizzazione di un circuito che permettesse di alimentarlo ed interfacciarlo correttamente con il resto del dispositivo.
La scheda progettata è riportata in Figura 23.

Figura 23: Scheda di comunicazione GSM/GPRS/GPS

In questa scheda sono stati riuniti il modulo *SIM808*, un regolatore di tensione configurato a $\tilde{4}.1$ V (tensione di lavoro ideale del modulo), un supporto per l'inserimento della scheda SIM dell'operatore, i connettori per le antenne e un buffer per adattare le tensioni sulla comunicazione Seriale.

7.2.2 Software

Per la maggior parte degli *AT Commands*, il modulo *SIM808* è compatibile con il codice *Arduino* preparato per il modulo *SIM908*.
Si hanno però delle differenze sui comandi legati alla localizzazione GPS.
Il nuovo modulo si basa sulla più internazionale *GNSS* (Global Navigation Satellite System).
Il *GNSS* è un sistema di satelliti che fornisce un posizionamento geo-spaziale autonomo con

copertura globale.

Come per tutte le altre funzioni, anche per la localizzazione attraverso *GNSS* si ricorre agli *AT Commands*.

Si riporta di seguito un esempio di implementazione su *Arduino* del comando *AT* che richiede al modulo *SIM808* la posizione:

```
// Invio del comando
Serial.println("AT+CGNSINF"); // Invio del comando AT Per richiesta poszione

// Ricezione della risposta
while(Serial.available()>0) { // Si verifica la presenza di comunicazione Seriale
    carattere = Serial.read(); // Si legge il carattere in arrivo
    posizione.concat(carattere); // Si compone la stringa ricevuta
}
```

Un esempio di risposta che si potrebbe trovare nella stringa posizione è:
+CGNSINF: 1,1,20160312134623.000,45.409084, 11.894066,171.800,15.35,273.6,1,,1.7,1.9,0.9,,11,5,,,27,,
Si osserva che ogni informazione è separata dalla vicina con una ",". Questo si presta all'elaborazione della stringa basata su tale carattere per l'estrazione delle informazioni necessarie. Le informazioni d'interesse sono longitudine e latitudine, che nel caso in esempio si può vedere facilmente che i valori sono 45.409084 e 11.894066.

7.3 Interfaccia dispositivi a bordo

Come fatto per *Arduino DUE*, si è provveduto a riprogettare anche la scheda di interfaccia per i dispositivi a bordo.

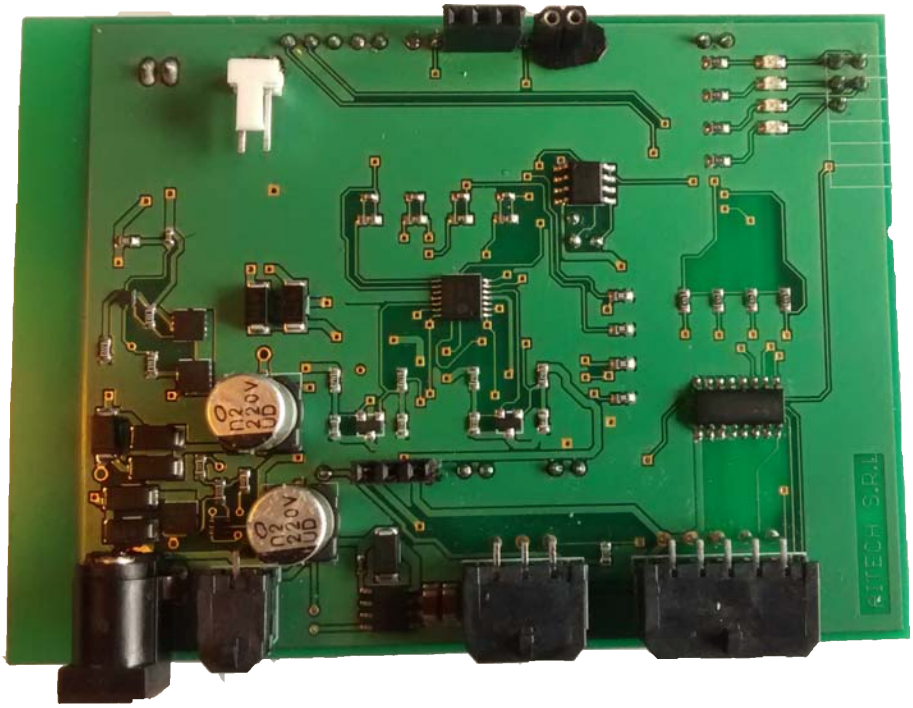


Figura 24: Interfaccia per dispositivi a bordo

Oltre al passaggio ai componenti SMD (necessario in un'ottica di risparmio di spazio), nella nuova scheda sono state apportate alcune modifiche rispetto a quanto fatto nella scheda assemblata a mano.

7.3.1 Aggiunta di LED

Nella nuova scheda si sono previsti quattro LED comandati da *Arduino* per un'interfaccia di debug:

- LED Giallo: Lampeggia quando è disponibile la localizzazione GPS
- LED Rosso: Attivo nella fase di inizializzazione del dispositivo
- LED Verde: Attivo nella fase di inizializzazione se corretto, lampeggiante durante il normale funzionamento
- LED Blu: Attivo quando ci sono delle richieste in sospeso che il modulo deve eseguire

In Figura 25 è riportato il classico schema di collegamento di un LED.

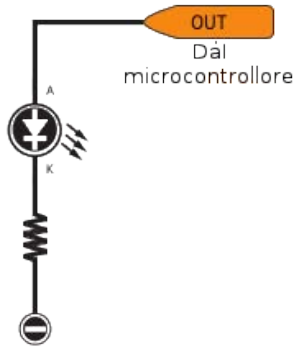


Figura 25: Schema di collegamento di un LED

7.3.2 Lettura delle tensioni

Nella nuova scheda si è scelto di modificare il sistema di lettura delle tensioni, optando per un sistema più robusto.

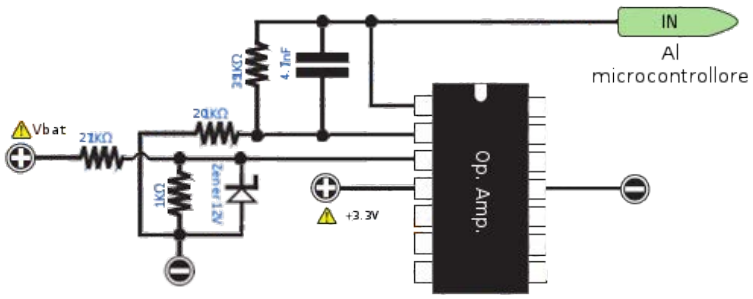


Figura 26: Schema lettura tensione batterie con amplificatore operazionale

Lo schema del sistema implementato è riportato in Figura 26. Si può notare che si usa ancora un partitore per ridurre la tensione in modo da rientrare nel range di lavoro di *Arduino*.

È stato aggiunto un diodo Zener in modo che un'eventuale sovratensione venga limitata. La tensione viene poi amplificata attraverso un amplificatore operazionale in configurazione non invertente, con tensione di alimentazione di 3.3 V. Si limita quindi la tensione di uscita a tale valore, garantendo così di non eccedere la tensione di lavoro del microcontrollore.

7.3.3 Aggiunta metodo di alimentazione

In corso d'opera è stato richiesto di aggiungere la possibilità di alimentare il dispositivo con un alimentatore. Infatti, per la maggior parte del tempo che l'imbarcazione è in darsena, questa ha a disposizione un collegamento alla rete elettrica.

Si è quindi aggiunto un jack al quale si può collegare direttamente tale alimentatore.
Per il corretto funzionamento, si è previsto un circuito di selezione dell'alimentazione. Questo è stato aggiunto perché si suppone che il cliente non scolleghi il connettore di alimentazione dalla batteria servizi prima di collegare l'alimentatore e viceversa. In assenza di un circuito di selezione dell'alimentazione, i problemi sono i seguenti:

- Se la tensione della batteria servizi supera la tensione dell'alimentatore, quest'ultimo diventa un utilizzatore attivo, con il rischio di danneggiamento.
- Se la tensione della batteria servizi è inferiore rispetto alla tensione dell'alimentatore, inizierebbe un processo di carica della batteria servizi, cosa che si vuole evitare soprattutto perché il sistema non è dimensionato in modo adeguato a supportare tale impiego.

Inoltre se l'utente ha a disposizione alimentazione dalla rete, non vuole che il dispositivo sia alimentato dalla batteria servizi, anche se questa dovesse avere una tensione superiore alla tensione dell'alimentatore.
Si è scelto un circuito di tipo analogico, perché il passaggio da un'alimentazione all'altra deve essere sufficientemente veloce per garantire che il microcontrollore non subisca un calo di tensione tale da causarne un reset.
Lo schema del circuito che soddisfa quanto detto è riportato in Figura 27.

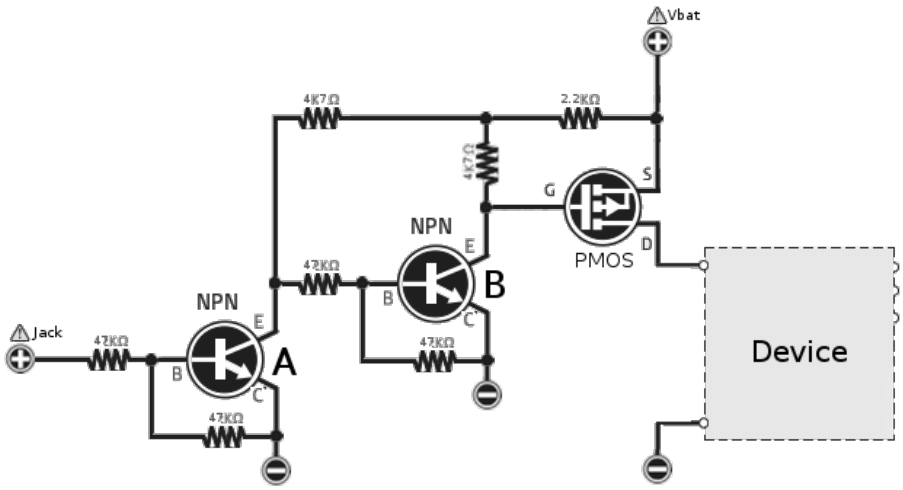


Figura 27: Schema circuito di selezione dell'alimentazione

Per semplicità, si consideri di modellizzare il comportamento dei transistor come contatti aperti o contatti chiusi. Si analizza il comportamento del circuito con le combinazioni di alimentazione possibili, tralasciano le combinazioni *"Jack Alimentazione" presente - "Batteria Servizi" assente* e *"Jack Alimentazione" assente - "Batteria Servizi" assente* perché prive di interesse.

- *"Jack Alimentazione" presente - "Batteria Servizi" presente*:
  Quando vi è tensione al "Jack Alimentazione", il transistor A risulta un contatto chiuso che porta la base del transistor B a GND. In tale situazione il transistor B risulta un contatto aperto, isolando il gate del P-MOS da GND, il quale si comporta come un contatto aperto a sua volta, interrompendo l'alimentazione del dispositivo da parte della "Batteria Servizi".

- *"Jack Alimentazione" presente - "Batteria Servizi" presente*:
  In questo caso si verifica l'inverso rispetto a quanto mostrato in precedenza: il transistor A (con base a GND) risulta un contatto aperto. La base del transistor B è quindi alimentata dalla "Batteria Servizi" stessa. In tale situazione il transistor B risulta un contatto chiuso che pone il gate del P-MOS a GND. Infine il P-MOS con tale configurazione è equivalente ad un contatto chiuso, consentendo l'alimentazione del dispositivo da parte della "Batteria Servizi".

8 Futuri Sviluppi

In conclusione vengono presentati gli sviluppi futuri di tale progetto:

- Implementazione del dispositivo in un'unica scheda elettronica: l'obbiettivo attualmente in fase di sviluppo è quello di riunire le tre schede che fino ad ora hanno composto il dispositivo descritto in questo documento, in una scheda elettronica unica, adatta alla produzione in serie, riducendo anche gli ingombri.

- Ampliamento delle comunicazioni: il potenziamento dei metodi di comunicazione, implementando anche la connettività bluetooth, già predisposta nel modulo *SIM808*, apre nuove strade per una comunicazione a costo zero e molto più rapida quando si è a bordo, consentendo quindi di implementare funzionalità nuove che permettano al cliente di effettuare sempre più azioni con il semplice tocco dello smartphone.

- Applicazione iOS: la creazione di un'app per sistemi operativi Android sicuramente non copre la totalità dei possibili clienti interessati ad eKEEPER2. Per questo motivo si è deciso di replicare, nell'immediato futuro, la stessa applicazione qui presentata per sistemi operativi Apple.

- Ampliamento servizi assistenza: ad oggi Aitech s.r.l. ha la possibilità di accedere alle diverse tabelle presenti nel server in modo da poter fornire assistenza ai clienti in caso di malfunzionamenti o di guasti. Verranno quindi sviluppati dei tools grafici i quali estrapoleranno determinate informazioni dai database e li presenteranno sotto forma di grafici temporali, fornendo così una "storia" dell'imbarcazione. I campi di maggiore interesse riguardano le batterie servizi, motore e tampone, le quali indicano lo stato di funzionamento dell'impianto elettrico complessivo della barca e di eKEEPER2.

- Replica delle funzionalità dell'applicazione su portale web: in linea con quanto operato da molti produttori di applicazioni (ad es. Facebook, Whatsapp,...), si vuole introdurre la possibilità di monitorare l'imbarcazione ed effettuare le operazioni descritte nelle sezioni precedenti anche tramite un apposito sito web. In questo modo anche utenti che non possiedono uno smartphone, oppure che non ne sono frequenti utilizzatori, possono facilmente interagire con eKEEPER2

- Adattamento del progetto eKEEPER2 al settore dei camper: il principio è simile a quello dell'imbarcazione, in cui l'utente è interessato a poter sorvegliare il proprio mezzo: anche in questo caso eKEEPER2 viene utilizzato come sistema di monitoraggio dei punti sensibili del camper (batterie, temperatura ambientale, intrusioni,...).

Riferimenti bibliografici

[1] Arduino. Arduino DUE.
URL: https://www.arduino.cc/en/Main/ArduinoBoardDue

[2] DFRobot. GPS/GPRS/GSM Shield V3.0.
URL: http://www.dfrobot.com/index.php?route=product/product&keyword=gsm%20gps&product_id=673#.VvKO5_nhBdg

[3] SIMCom. SIM808.
URL: http://simcomm2m.com/En/module/detail.aspx?id=137

[4] Arduino. Language Reference.
URL: https://www.arduino.cc/en/Reference/HomePage

[5] Wikipedia. Finite-state machine.
URL: https://en.wikipedia.org/wiki/Finite-state_machine

[6] Wikipedia. UART.
URL: https://it.wikipedia.org/wiki/UART

[7] Wikipedia. NMEA 0183.
URL: https://en.wikipedia.org/wiki/NMEA_0183

[8] Processing. HomePage.
URL: https://processing.org/

[9] Matteo Conti. Sistema di Monitoraggio da Remoto per Imbarcazioni: Interfaccia Utente.